Data Providers' Handbook

Archiving Guide to the PDS4 Data Standards

DRAFT

Data Design Working Group

28 May 2012

Version 0.3.8

CHANGE LOG

Revision	Date	Description	Author
0.1	Mar 30, 2009	Initial draft based on information collected by the Data System Working Group.	R.Joyner
0.2	Aug 6, 2009	Updated versions of all Classes	R. Joyner
0.2.1	2010-08-31	Complete overhaul, but only partly successful through page 25	R. Simpson
0.22	Aug 31, 2010	Integrate Simpson and Joyner docs	R. Joyner etal
0.22.1	2010-09-22	Edits through Section 3, except Section 2	Simpson
0.22.2	2010-09-24	Added comments from Mitch Gordon	Simpson
0.22.2	2010-10-01	Significant edits to Section 4	Simpson
0.22.3	2010-10-25	Significant edits to Sections 1,3-6	Simpson
0.3	2011-04-15	Significant edits addressing Build 1b comments	M.Gordon etal
0.3.1	2011-04-21	Additional content added	M.Gordon etal
0.3.2	2011-05-02	Significant reorganization, additional edits	M.Gordon etal
0.3.3	2011-06-29	Make Sections 1-2 more user friendly	R. Simpson, M. Gordon
0.3.4	2011-09-06	Major changes Sections 2, 3, 8,14	M. Gordon, R. Joyner
0.3.5	2011-10-31	Updates for schema 0.5.00g	M. Gordon, R. Joyner
0.3.6	2012-01-31	Updates for schema 0.7.0.0.j	M. Gordon, R. Joyner
0.3.7	2012-03-28	Updates for schema 0.7.0.0.j	M. Gordon, R. Joyner
0.3.8	2012-05-28	Updates for schema 0.8.0.0.k	M. Gordon, R. Joyner

TABLE OF CONTENTS

1.0 INTRODUCTION

Planetary Data System version 4 (PDS4) represents a departure from previous versions of the PDS. Although it is still an archive of planetary data, it has been designed using contemporary information technology concepts and tools. The system is built around a 'data model' that rigorously defines each of its components and the relationships among them. There are only four fundamental data structures, but many extensions are possible — each also rigorously defined. By carefully controlling product definitions and relationships, PDS can accurately track the progress of each product entering the system, compute detailed inventories of holdings, design sophisticated services that users can request to act on subsets of the archive (such as transformations and displays, in addition to the expected search and retrieval functions), and connect data products to relevant internal and external information (documentation).

Note that this version of the document is conforment to version "0.8.0.0.k" of the Information Model.

1.1 Purpose

The *Data Providers Handbook* (*DPH*) is a guide for preparation of data being submitted to PDS4. It will walk you through preparation of very simple products, collections of products, and bundles of collections, which are the units in which deliveries are made to PDS4.

1.2 Audience

The *DPH* is written for scientists and engineers in the planetary science community who are planning to submit new or restored data to PDS4 (data providers). While the document is applicable to all such submissions, most of the examples and discussions are presented in a mission/instrument context.

1.3 Reader Preparation

The *DPH* is one of several documents describing the PDS4 system.

Readers, even those very familiar with previous versions of PDS, **should read the *PDS4 Data Standards Concepts Document* [4] and the *Glossary of PDS4 Terms* [5] before beginning the Data Provider's Handbook.**

The *DPH* should be used in conjunction with the *PDS Standards Reference* (PDSSR) [2] and the *PDS4 Data Dictionary* [3a,b], which have been updated for PDS4. For more information on the individual documents, see the *Introduction to the PDS4 Document Set* [7].

1.4 XML Editors

PDS4 is implemented in the eXtensible Markup Language (XML), a set of 'open source' rules for encoding documents and data structures in machine-readable form with special applicability to providing web services. It is beyond the scope of the *DPH* to include an XML tutorial; however, Appendix B "Reading a PDS4 XML Schema" provides some introductory information.

The use of an XML editor simplifies construction and validation of schemata (the plural of 'schema') and labels. Two editors have been popular during development of PDS4.

oXygen	**http://www.oxygenxml.com**	which is licensed software, and
Eclipse	**http://www.eclipse.org/**	which is open source software.

1.5 Document Notation

A few words have meanings which differ depending on the community in which they are used. We have adopted modifiers to help distinguish among multiple uses. For example, 'attribute' is widely used in both PDS and XML — but its meaning in each case is different. In this document we use 'PDS attribute' and 'XML attribute' to establish the context.

1.6 Applicable Documents

1.6.1 Controlling Document

[1] Planetary Data System (PDS) PDS4 Information Model Specification, Version 0.8.0.0.k

1.6.2 Reference Documents

[2] Planetary Data System Standards Reference, JPL D-7669, Part 2

[3a] PDS4 Data Dictionary - Abridged - V.0.8.0.0.k.

[3b] PDS4 Data Dictionary - Unabridged - V.0.8.0.0.k.

[4] PDS4 Data Standards Concepts, September 12, 2011.

[5] Glossary of PDS4 Terms, October, 2011, Version 2011-10-28 (v111028).

[6] PDS4 Data Dictionary Tutorial, October 28, 2011.

[7] Introduction to the PDS4 Document Set, October 30, 2011.

1.7 Other Resources

1.7.1 XML Schema location

http://pds.nasa.gov/pds4/schema/released/pds/

1.7.2 PDS4 Software

http://pds.nasa.gov/pds4/software/validate/

PART I. PRELIMINARIES

2.0 BUILDING BLOCKS

2.1 Terminology

The results of our exploration of the Solar System can be loosely described as *data objects*; these can be electronic files, dust samples, or a sense of awe at the wonder of the universe. For purposes of archiving, we need a *description* to accompany each data object — in the case of an electronic file, a digital object, we need both the structure and meaning of the file contents for it to be useful. We can't fit dust samples or senses of awe into PDS4, but we can fit their descriptions. A description paired with its data object (when available — *e.g.*, if a digital object) is called an *information object*. If many data objects have similar characteristics, we can group them into a *class* and the common characteristics are the defining *attributes* of that class.

A *product* is one or more closely related information objects for which the descriptions have been combined into a single XML *label* and for which the product has a PDS-unique *identifier*. Closely related products may be grouped into a *collection*; in fact, every product entering PDS must be a member of some collection. Closely related collections may be grouped in a *bundle*.

For example, a planetary image, the histogram of its pixel values, and the descriptions of both could be organized as a product. Many such products — perhaps of the same target — could be defined as a collection. Image collections from many targets along with appropriate documentation, calibration, etc. (separate collections) could be a bundle, which would be a deliverable to PDS.

For more rigorous definitions of the terms introduced above, see the *Glossary of PDS4 Terms* [5].

2.2 Data

A word of caution about terminology – we have tried to avoid using differently terms that have strong PDS3 connotations. Unfortunately the English language does not provide a sufficient set of meaningful, unique, unambiguous terms to meet all of our needs. Please do not rely on the names of things – review carefully the PDS4 definitions.

Recall from the PDS4 Glossary, a "Product" is one or more closely related information objects grouped together and having a single PDS-unique identifier. In the PDS4 implementation, the descriptions for all of the objects making the product are combined into a single XML label.

Four basic structural data formats are allowed in PDS4.

1. repeating_record_structure

- Suitable for fixed length tabular data.
- May be either binary or character, but a single object may be only one of these.
- The records are fixed length.
- The elements of a record (fields) are heterogeneous.
- The formats (data type and size) of fields in the same position in each record are the same (i.e., the second field in the second record is constructed identically to the second field in the first record).

2. homogeneous_array_structure
    - Suitable for images, spectra, spectral cubes, maps, etc.
    - The elements of an array are homogeneous.
    - The individual elements of any array are stored with their bytes in the order dictated by their scalar type.
    - PDS requires a specific order in which the elements of the array are stored. The order is described in the Concepts Document, Section 9.3, and the Standards Reference, Section 5.2.

The majority of PDS4 objects can be supported by the two previous structures. For those PDS4 objects which can not supported by the above, we have two additional stuctures distinguished by whether or not software must be used to decode the information before it can be accessed for reading, display, or analysis.

3. parsable_structure
    - Suitable for plain text, HTML, XML, tabular data with variable length fields and records (delimited text).
    - The contents are a byte stream which can be parsed with standard rules (e.g., comma separated entries, standard punctuation);t no decoding software (e.g., Adobe Acrobat©) is required.

4. encoded_structure
    - Suitable for documents, browse products, etc., but not generally not for observational products.
    - Contents are a byte stream that must be decoded by software before use.
    - The use of encoded_byte_stream objects is restricted by PDS to a limited set of PDS approved external standards (e.g., PDF-A, JPEG, GIF).
    - Only in exceptional cases will encoded_byte_stream objects be considered appropriate for storing observational data. Prior PDS approval is required.

Each of these structural formats corresponds to one PDS4 "base class" and each PDS4 "base class" uses one and only one of the structural formats.

Structural Format		Base Class
• repeating_record_structure	↔	Table_Base
• homogeneous_array_structure	↔	Array_Base

• parsable_structure	↔	Parsable_Byte_Stream_Base
• encoded_structure	↔	Encoded_Byte_Stream_Base

Here are a few rules (for a full description, see the Standards Reference, Section 5):

- Each digital object must be stored in one of the four basic data formats.
- A digital object must be contained in a single file (i.e., a digital object cannot span multiple files)
- A file may contain multiple digital objects from the same product.
- Digital objects within a file are not required to use the same storage structure.
    - Can have a header (parsable structure) and an image (array structure) in the same file.
- When multiple digital objects are contained in a file, they must be stored sequentially (i.e., readout of the file should be sequential where one object is followed by the next, not two objects interleaved).

These data structures only tell you how to read the bytes from a file; they contain absolutely no interpretation beyond that. PDS4 defines a set of base object classes that apply the first level of interpretation to the bytes read (the "It's an image" interpretation), and subclass extensions of those base object classes to expand and restrict the associated metadata for specific object types.

2.2.1 Sample Data

This document frequently references a set of example PDS4 products – see Section 14 - Example PDS4 Products.

There are two different sets of examples:

- The first is a set of example products. This includes a representative set of products that would exist within an archive (e.g., character table, binary table, document, etc.).

- The second set is an example of a complete archive. This includes all products that would comprise a complete PDS4 archive (e.g., bundle, collections, collection inventories, aareadme, errata, and basic products).

2.2.2 Methodology

To complete the sample data (from above), we created XML labels for each of the products in the archive. For simplicity, we assumed that each product is an object-description pair. The products are grouped into collections. The collections are grouped into a single “archive” bundle, and the bundle was delivered to PDS.

3.0 SCHEMA AND LABELS – AN OVERVIEW

Label development begins with an XML schema. PDS maintains a master-schema that serves as a library of generic XML schema for each product type based on the core portion of the PDS Information Model (IM).

In consultation with your PDS discipline node (DN), for each product type, determine which schema you will need. The DN will use the master-schema, possibly incorporate both node and mission specific schemas appropriate for the products you plan to produce, and pass a 'tailored' version of the schemas to you.

As the data provider, you might make minor modifications to further "tailor" the node and/or mission schema to produce a schema that more closely describes your data. For example, your editing might include changing some classes and attributes from' optional' to 'required', or inserting additional node and possibly mission specific classes appropriate for the products you plan to produce.

Any and all changes that you might make to your tailored schema must be validated against the schema you received from the DN which, in turn, will have been validated against the generic schema on which each was based.

Finished labels must be validated against the tailored and generic schema on which they are based.

3.1 Pipeline Considerations

If you are developing a collection with more than a few products, you will want to automate label generation as much as possible.

There are at least two approaches:

a) Use a schema (xsd file) as input to the pipeline software you use to generate your labels, or

b) Use a label template (xml file) as input to the pipeline software. One feature of many XML editors is the ability to generate such a template.

The template looks like the final label except that, depending on how you set some options, there are either no values between the XML tags, or the values which vary from one label to the next are represented by placeholders which the pipeline software will replace.

As with everything, there are advantages and disadvantages to each approach. The first consideration will probably be the software package underlying your pipeline and whether or not it is specifically designed to handle XML.

In principle, you should make as many of your edits as possible in the schema to provide yourself and PDS the most robust criteria against which to validate the individual XML label files produced for your archive.

In this handbook, we assume the pipeline will use an XML template. This provides a framework for the discussion. Even if your pipeline will use the XML schema as input, you will still need to read the sections on ‘Schema Editing’ including the section on ‘Label Template Editing’.

In this handbook, we use Product_Table_Character for most of our examples. The tailored schema you receive from your node may differ slightly from the ones in these examples. Our goal here is to become familiar with the contents and to understand the process.

3.2 Permitted Schema Modifications

When you modify a schema, the resulting schema must still satisfy its parent schema’s restrictions. In principle this means you can modify the schema to make it more restrictive, and you may also add classes within certain constraints.

The master-schema should not be modified. This schema is to be considered “read-only” and the content is immutable. If you think you have found a reason to modify this schema, contact your discripline node as something has gone terribly wrong.

The node and mission schemas are subject to modification. Your discipline node will make a best effort attempt to create node and mission schemas appropriate for the products you plan to produce. But, as stated above, you might find that you are able to make modifications that will more closely fit the particular nature of the observational and supplementary products in the archive.

The following is a summary of the type of modifications that you, as the data provider, might find necessary to make:

a) You may change an XML element from optional (minOccurs=”0”) to required (minOccurs=”1”, or some number that is less than or equal to the value specified in maxOccurs).
b) You may delete an optional XML element (if minOccurs=”0”)
c) You may restrict the upper limit on the number of times the XML element will be used by setting maxOccurs to a value that is greater than or equal to the value specified in minOccurs.
d) For those XML elements which have “maxOccurs = unbounded”, you may set an explicit upper limit on the number of times the XML element will be used.

e) If an XML element will have the same value for all products being produced from this schema you may insert that value into the XML element in the schema.
f) During the initial tailoring, the PDS node staff may insert additional classes in either the Mission Area and / or the Node Area (subsections within the Observation_Area).
g) The data provider may insert additional classes in the Mission Area (a subsection within the Observation_Area).

You may not modify the minOccurs or maxOccurs attributes such that they would specify a less restrictive set of conditions. For example, if "minOccurs = 5" you cannot set "minOccurs = 4", as this would violate the initial restriction. Similarly, if "maxOccurs = 1", you cannot set "maxOccurs = 2".

There are really three options for what you will do in your schema with XML elements which are 'optional' in the parent schema:

1. The XML element will be used across all labels
2. The XML element will sometimes be used in one or more labels.
3. The XML element will not be used in any labels

There are several approaches to handling optional XML elements based at least in part on your label pipeline design. We recommend:

- Set "minOccurs = 1" (or some appropriate higher value) for the optional XML elements you intend to use in every label.
- Delete the optional XML elements that will not be used in any label (if "minOccurs = 0").
- Leave "minOccurs = 0" for the optional XML elements you intend to use only for some subset of the products. When maxOccurs is unbounded, reset it to an appropriate upper value. Note maxOccurs must be greater than or equal to minOccurs.

3.3 Developing Local Data Dictionaries

Local data dictionaries (e.g., mission and node dictionaries) are developed by the data preparer in conjunction with discussions and review by the lead PDS discipline node. Local dictionaries contain new classes and attributes as needed to provide detailed product descriptions which can not be defined using the existing PDS classes and attributes

A local dictionary is created when the data preparer needs to define attributes and classes specific to his mission, observing campaign, data restoration effort, etc, including but not limited to specific instrument parameters, and additional observational parameters. They are an essential tool for providing data preparer's latitude to tailor their labels to more closely fit the particular nature of the observational and supplementary data in the archive.

Local data dictionaries are discussed in more detail in Section 11.

PART II. THE FIRST STEPS

4.0 OUTLINE THE BUNDLE

We introduce the ‘Voyager 2 Jupiter Encounter Data’ archive that was originally produced by the PPI node and delivered to the PDS as a PDS3 dataset. This will provide the basis for all of the specific discussions which follow.

Our spacecraft is the ‘Voyager 2’ (VG2) spacecraft and the instrument is the ‘Plasma Science Experiment’ (PLS) a plasma instrument designed to detect plasma conditions throughout the Voyager trajectory.

	Name	*Abbreviation / Acronym*
Spacecraft	Voyager 2	VG2
Instrument	Plasma Science Experiment	PLS

4.1 Collections in the Bundle

Refer to the Standards Reference and confer with your PDS discipline node to determine all of the required and appropriate collections for your bundles.

Using the archive from our sample data, the instrument team plans to submit a single “archive” bundle to PDS, for the Venus encounter data. The bundle will consist of five collections.

Bundle:
- Browse Collection
- Context Collection
- Data Collection
- Document Collection
- XML Schema Collection

4.2 Directory Organization

Our sample “archive” bundle is a fairly simple bundle that uses a fairly simple directory structure. Since there are only a small number of collections, we elect to have one directory in the bundle root for each collection.

The bundle root must contain at least one file, the XML label file for the bundle product, and may only contain one additional file – an optional readme file which if used is described in the bundle XML label file.

```
bundle root
| - bundle.xml
|
| - browse
|    | - collection_browse.xml
|    | - collection_browse_inventory.tab
|    | - collection product1
|
| - context
|    | - collection_context.xml
|    | - collection_context_inventory.tab
|
|    | - context_product1
|    | - context_product2
|    | - context_product3
|
| - data
|    | - collection_data.xml
|    | - collection_data_inventory.tab
|    |
|    | - data product1
|
| - document
|    | - collection_document.xml
|    | - collection_document_inventory.tab
|    |
|    | - doc_product1
|    | - doc_product2
|    | - doc_product3
|
| - schemas
|    | - collection_xml_schema.xml
|    | - collection_xml_schema_inventory.tab
|    |
|    | - xml_schema_product1
|    | - xml_schema_product2
|    | - xml_schema_product3
```

The root level subdirectories each correspond to a single collection. Each directory will contain the collection XML label file and the collection inventory file. Depending on the size of the collection, it may or may not contain other files.

In our example data, there is a single file in the data directory, ELEMON.TAB.

However, had there been multiple observational data files, it would have been reasonable for the team to decide to use the spacecraft clock count at the start of each observation as the primary reference for each observation. This would have been used as the filename root and in the Logical Identifier (LID) for each observational data product. There are several approaches to setting names for each observational product and the corresponding subdirectories.

4.3 Determine the Documentation Needed

Refer to the Standards Reference and confer with your PDS discipline node to determine all of the required and appropriate documentation for your document collection(s).

You and the consulting PDS node should agree on a list of required documentation early in the design process. Documentation considered essential to understanding or using the archive and the underlying data in the archive, except for published journal articles, must be submitted as part of the archive. Journal articles may be included if permitted by the copyright holder. Each document to be archived must be prepared and saved in a PDS-compliant format, Refer to Section 11 of the Standards for a list of PDS approved formats.

In our archive example, documentation includes the following documents:

1. An errata file that describes any changes or errors in the archive.
2. A copy of a published journal article that describes the mission (in both ASCII and HTML).
3. A copy of a published journal article that describes the instrument (in both ASCII and HTML).
4. A checksum file that lists the MD5 checksum of the files in the archive. Note that this file is not a required PDS4 document. It is included in the PDS4 archive simply because it was part of the original PDS3 data_set.

Each of the above document products is individually labeled. Both the errata and the checksum files were each labeled using the Product_File_Text schema as these documents are strictly ASCII text. The other two document products, the mission and instrument descriptions, were each labeled using the Product_Document schema as these documents are presented in both an ASCII and an HTML version. Note that the two forms of the document, the ASCII and HTML versions, are collectively a single document product (i.e., All versions of a document are considered part of a single PDS document product).

For information on how to populate the attributes and classes in a Product_Document, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

5.0 DESIGN LOGICAL AND VERSION IDENTIFIERS

Every product label contains an identifier which must be unique across all products archived with the PDS. This identifier is referred to as a LIDVID and is the concatenation of a Logical Identifier (LID) and a version identifier (VID). We'll address the construction of each in the following sections.

5.1 General Concepts

Here are some general rules:

- LIDs must be unique across PDS
- Each PDS4 LID is constructed as a Uniform Resource Name (URN)
- Each LID in a PDS archive begins with 'urn:nasa:pds'
- LIDs are case insensitive.
- LIDs are restricted to ASCII letters and numbers, dash, underscore, and period. Colons are also used but only in a very prescribed way discussed in this section.
- LID maximum length is 255 characters.

The complete set of requirements for LID construction is given in Section 7 of the PDS4 Standards Reference.

5.2 Constructing LIDs

Detailed requirements and formation rules are provided in the PDS Standards Reference, Section 7.1; we provide a brief summary here.

Recall that each basic product is delivered to PDS as a member of a collection, and that collection is a member of a bundle. LIDs are constructed based on a hierarchical set of relationships.

We can think of LIDs as constructed by concatenating fields of characters. The fields are separated by colons. This is the only use of colons permitted in LIDs.

- Bundle LIDs -- are constructed by appending a bundle specific field to 'urn:nasa:pds'.

  Bundle LID = urn:nasa:pds:<bundle field>

  Since all PDS bundle LIDs are constructed this way, the bundle field must be unique across all products archived with the PDS.

- Collection LIDs -- are constructed by appending a collection field to the parent bundle's LID.

Collection LID = urn:nasa:pds:<bundle field>:<collection field>

Since the collection LID is based on the bundle LID, which is unique across PDS, the only additional condition is that the collection field must be unique across the bundle.

- Basic Product LIDs -- are constructed by appending a product field to the parent collection's LID.

  Product LID =
  urn:nasa:pds:<bundle field>:<collection field>:<product field>

  Since the product LID is based on the collection LID, which is unique across PDS, the only additional condition is that the product field must be unique across the collection.

5.2.1 Examples

The following examples are based on a hypothetical mission.

	Name	***abbreviation***
spacecraft	Super SpaceCraft 01	ssc01
instrument	High Resolution Photon Counter	HiResPC
cruise phase	Cruise, Mercury, Earth phase	Cruise

The team decides to use the spacecraft clock count at the start of each observation as the product field of the LID for observational data products.

This is all the information we need to start designing LIDs.

Cruise Phase

Bundle

urn:nasa:pds:ssc01.HiResPC.Cruise

Collections

urn:nasa:pds:ssc01.HiResPC.Cruise:Browse
urn:nasa:pds:ssc01.HiResPC. Cruise:Context
urn:nasa:pds:ssc01.HiResPC. Cruise:Data
urn:nasa:pds:ssc01.HiResPC. Cruise:Document
urn:nasa:pds:ssc01.HiResPC. Cruise:Schema

Products [data products for sclock = 31234567]

urn:nasa:pds:ssc01.HiResPC. Cruise:Browse:browse_31234567
urn:nasa:pds:ssc01.HiResPC. Cruise:Data:data_raw_31234567
urn:nasa:pds:ssc01.HiResPC. Cruise:Data:data_derived_31234567
urn:nasa:pds:ssc01.HiResPC. Cruise:Document:errata
urn:nasa:pds:ssc01.HiResPC. Cruise:schema:Table_Character_0411f

5.3 VID Construction

Detailed requirements and formation rules are provided in the PDS Standards Reference, Section 7.2; we provide a brief summary here.

Version IDs are used for all types of products, including basic products, collections, and bundles.

- Version IDs must be of the form M.m where "M" and "m" are both integers. "M" is the "major" component of the version and "m" is the "minor" component of the version.
- The major number is initialized to one for archive products. (Zero may be used for sample products or test run products that are not intended for the archive.) The minor number is initialized to zero.

The VIDs in all of the products in our sample archive (e.g., bundle, collection, and product) is 1.0 which corresponds to the first submission for the archive (i.e., this version is the post peer-review, lien resolved version which is the first version that will be archive with the PDS and will go into the deep archive). Preliminary versions may not use version 1.0.

5.4 LIDVID Construction

Concatenate the LID and VID using a double colon as the connector. Here are sample LIDVIDs based the example LIDs in Section 5.2.1

5.4.1 Examples

Cruise Phase
Bundle
urn:nasa:pds:ssc01.HiResPC.Cruise::1.0

Collections
urn:nasa:pds:ssc01.HiResPC.Cruise:Browse::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:Context::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:Data::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:Document::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:Schema::1.0

Products [data products for sclock = 31234567]

urn:nasa:pds:ssc01.HiResPC. Cruise:Browse:browse_31234567::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:Data:data_raw_31234567::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:Data:data_derived_31234567::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:Document:errata::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:schema:Table_Character_0411f::1.0

5.5 LIDVIDs – The Next Step

LIDs and LIDVIDs will be ubiquitous in your archive. Be sure you have the naming convention / formation rule / algorithm correct before proceeding. When you have constructed draft LIDs and LIDVIDs contact your PDS DN to verify they are unique and conform to the requirements.

Once you and your DN have settled on a naming convention / formation rule for applying LIDs and LIDVIDs to the various products in your archive, the next step is to apply the formation rule to your pipeline that automatically generates the labels in your archive.

PART III. BASIC PRODUCT LABELS

6.0 BASIC PRODUCT LABELS - AN OVERVIEW

The labels for basic products (all products except Collection and Bundle products) structurally fall into two groups: observational products and all other basic products. We start with the former; then we will discuss the aspects of non-observational product labels which differ.

6.1 Basic Product Labels – Observational Products

6.1.1 Selecting the Appropriate Product Schema

The data provider and a representative from the DN discuss the anticipated nature of the observational data. Once the DN has sufficient information, the DN data engineer will select the appropriate schema for each product type, make modifications as appropriate to generate tailored schemas, and then pass the schemas to you, the data provider. Currently, there is a single generic "Observational" product schema from which the DN will choose from 13 possible product types:

For binary array data:

- Array_2D_Image
- Array_2D_Map
- Array_2D_Spectrum
- Array_3D_Image
- Array_3D_Movie
- Array_3D_Spectrum

For binary tabular data:

- Table_Binary

For character tabular data:

- Table_Character
- Table_Delimited

For "other" data:

- Header
- Header_Encoded
- Parsable_Byte_Stream
- Stream_Text

Any basic product label may support multiple objects and multiple object types. For example, the generic Product_Observational schema can include the description of an Array_2D_Image and a Table_Character histogram of pixel values. Both object types form the single digital product.

The hook to incorporating additional allowed object types is via the File_Area_Observational class which permits the inclusion of any additional digital objects (e.g., Header, Table_Character, etc) that are specified in connection with the the Array_2D_Image as a single digital product. For additional information, consult your DN.

6.1.2 Basic Product Label Organization

All PDS XML labels begin with a section of statements known as the XML preamble. Here is an example of these statements:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation= "http://pds.nasa.gov/schema/pds4/pds/v08
http://pds.jpl.nasa.gov/pds4/schemas/PDS4_PDS_0800k.xsd">
```

For more information about the XML preamble contact your discipline node. Additional information about the XML preamble can be found on the PDS Wiki:

https://oodt.jpl.nasa.gov/wiki/display/pdscollaboration/XML+Catalogs

In a typical basic product label there are a set number of major blocks of information (areas). Each of these areas contains one or more classes each of which may have several attributes. The areas are:

- XML Preamble
- Identification Area
    - Provides identification information for the product.
- Observation Area
    - Provides classes describing the specific observation, laboratory experiment, etc.
    - Includes subsections for relevant classes specific to one or more discipline nodes, and to the mission (or equivalent name space).
- Reference List Area
    - Provides identification information for products, journal articles, etc., relevant to understanding the product. References are made to sources both internal and external to PDS.

- File Area
    - Identifies the file(s) containing the data object(s), and
    - Provides classes specific to each data object in a given file (e.g., the description and parameters of each header, table, image).

Figure 6-1 depicts the major blocks of information (areas) in a typical basic product label.

```
<?xml version="1.0" encoding="UTF-8"?>
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation= "http://pds.nasa.gov/schema/pds4/pds/v08
     http://pds.jpl.nasa.gov/pds4/schemas/PDS4_PDS_0800k.xsd">

     <Identification_Area>      27 lines
     <Observation_Area>         76 lines
     <Reference_List>           10 lines
     <File_Area_Observational> 41 lines
</Product_Observational>
```

Figure 6-1. Example Class organization for Product_Observational

All of the basic products adhere to the structure identified in Figure 6-1. This structural consistency enables us to look at the construction of the label for a single basic product type in detail with the knowledge that the same techniques will be applicable across all label types. This structural consistency also simplifies data management and user browsing.

We will use Product_Table_Character for the following discussion.

Note that Figure 6-1only reflects the content of the labels with respect to the major classes / areas and the underlying hierarchy within the areas. There is a significant amount of content that is not depicted including the attributes contained in the classes.

For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of a basic product label, consult the PDS4 Data Dictionary. For information on how to populate the attributes and classes, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

6.2 Basic Product Label Organization – Non-Observational Products

The labels for the non-observational products are structurally similar to that of the observational products. This section highlights those differences.

6.2.1 Selecting the Appropriate Product Schema

The process for selecting the appropriate schema for the non-observational products is the same as that used for the observational products.

The data provider and a representative from the DN discuss the various types of ancillary data that will be supplied as part of the archive (e.g., documents, browse, thumbnails, etc). Once the DN has sufficient information, the DN data engineer will select the appropriate schema for each product type, make modifications as appropriate to generate tailored schemas, and then pass the schemas to you, the data provider. The DN will choose from the following possible product types:

- Product_Browse
- Product_Document
- Product_Thumbnail

Unlike observational products which can support multiple object types, non-observational product label only support a single object type. For example, the Product_Document product may only describe a single digital document product.

6.2.2 Basic Product Label Organization

The organization of a non-observation product label is fairly well aligned to the observational product label. The PDS XML labels begin with a section of statements known as the XML preamble. There is a difference in the preamble of a non-observational product in that non-observational products will not reference local dictionaries. Therefore the preamble only describes the reference to the master-schema. Here is an example of these statements:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product_Document xmlns=“http://pds.nasa.gov/pds4/pds/v08”
  xmlns:pds=“http://pds.nasa.gov/pds4/pds/v08”
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation= "http://pds.nasa.gov/schema/pds4/pds/v08
http://pds.jpl.nasa.gov/repository/pds4/schemas/PDS4_PDS_0800k.xsd">
```

For more information about the XML preamble contact your discipline node. Additional information about the XML preamble can be found on the PDS Wiki:

https://oodt.jpl.nasa.gov/wiki/display/pdscollaboration/XML+Catalogs

In a typical non-observational product label there are a set number of major blocks of information (areas). Each of these areas contains one or more classes each of which may have several attributes. The areas that are shared by all of the non-observational products include:

- XML Preamble
- Identification Area
    - Provides identification information for the product.

- Reference List Area
    - Provides identification information for products, journal articles, etc., relevant to understanding the product. References are made to sources both internal and external to PDS.

The following is a partial list of areas that are specific to one or more of the non-observational products:

- File Area_Browse
    - Identifies the file(s) containing the data object(s), and
    - Provides classes specific to each data object in a given file (e.g., the description and parameters of each browse product).

- File Area_Encoded_Image
    - Identifies the file(s) containing the thumbnail data object(s), and
    - Provides classes specific to each data object in a given file (e.g., the description and parameters of each thumbnail product).

- Document_Format_Set
    - Identifies the set of data objects that collectively form the document.
    - Provides identification information specific to each type of document format (e.g., html, pdf).

- Document_Description
    - Provides descriptive information about the document being described.

Note that the above only reflects the content of the labels with respect to the major classes / areas and the underlying hierarchy within the areas. There is a significant amount of content that is not depicted including the attributes contained in the classes.

For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of a basic product label, consult the PDS4 Data Dictionary. For information on how to populate the attributes and classes of either a Document or Browse product, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

7.0 LABEL TEMPLATE EDITING

The XML label template, as the name implies, is an xml file that will be used as a template for generating / producing the actual XML label(s) that will be used in your archive.

We can use the tools provided in an XML aware editor to generate a label template from the master-schema (xsd). You will generate a label template for each type of product that will be used in your archive (bundle, collection, and each type of basic product).

The following is a list of the types of label templates that can exist in your archive:

- Product_Browse
- Product_Bundle
- Product_Collection
- Product_Context
- Product_Document
- Product_File_Text
- Product_Observational
- Product_SPICE_Kernel
- Product_Thumbnail
- Product_XML_Schema

7.1 Creating a Label Template

The XML label template is an xml file which looks much more like the final label than does the schema. The label template is created from the master-schema (xsd) using your favorite xml-aware editor (XAE). The steps for creating the label template are as follows:

Steps:
1. Download a copy of the current masterSchema (xsd)
2. Open the masterSchema in xml-aware editor (XAE)
3. Customize the settings
4. Create label template (xml)
5. Ensure label template is valid

For additional information on how to create a label-template can be found in Appendix E – Steps to Create a Label Template..

7.2 Label Editing

In the label-template, values which vary from one label to the next will be represented by placeholders that either you or the pipeline software will replace. In the examples in this and following sections, we show fragments of XML templates. These generally contain dummy values inserted by the XML editor, for example name1 in <name>name1</name>.

Values which are identical across all labels may be set to their appropriate value in the template.

In terms of making changes / editing the label template, there aren't that many types of modifications that can be made:

1. You can remove, from the label template, the classes or attributes designated as optional in the parent schema.
2. You can modify the number of times a class or attribute is repeated
3. You can set the value of an attribute to be fixed / static
4. You can specify a value from an enumerated list of values

Keep in mind that all changes / modifications to the label template must adhere to the constraints dictated by the referenced schema and schematron.

We will address each of these types of modifications in the sections that follow.

7.2.1 Label Editing - Presence / Absence of a Class or Attribute

Recall that in a schema you indicate the presence and / or absence of a class or attribute by setting the minOccurs and maxOccurs values appropriately (i.e., Setting "minOccurs=0" and "maxOccurs=0", the XML element is not expected to be present in the label template).

You have the same control over the presence and absence of classes and attributes within the label template. Use either your favorite text editor or XML aware editor to edit the label template and simply add or delete the class or attribute from the label template. Note that using an XML aware editor allows you to validate in real time as you make your edits.

7.2.2 Label Editing – Repeating a Class or Attribute

Recall that in a schema you may indicate the number of times a class or attribute will repeat by setting the minOccurs and maxOccurs values appropriately. For example, if the schema specifies that there are 5, and always 5, fields in one row of a table (i.e., "minOccurs=5" and "maxOccurs=5"), you will need to simply replicate the rows of the table such that there are 5 instances.

```
<Record_Character>
     <record_length unit="byte">50</record_length>
     <fields>50</fields>
     <Field_Character>
         <name>name6</name>
         <field_number>50</field_number>
         <field_location unit="byte">50</field_location>
         <data_type>data_type0</data_type>
         <field_length unit="byte">50</field_length>
         <field_format>field_format0</field_format>
         <unit>unit1</unit>
         <scaling_factor>0</scaling_factor>
         <value_offset>0</value_offset>
         <description>description11</description>
     </Field_Character>
          .
          . <add 3 more instances here>
          .
     <Field_Character>
         <name>name6</name>
         <field_number>50</field_number>
         <field_location unit="byte">50</field_location>
         <data_type>data_type0</data_type>
         <field_length unit="byte">50</field_length>
         <field_format>field_format0</field_format>
         <unit>unit1</unit>
         <scaling_factor>0</scaling_factor>
         <value_offset>0</value_offset>
         <description>description11</description>
     </Field_Character>
 </Record_Character>
```

Repeat classes and attributes within the label template must adhere to the restrictions specified in the reference schema.

7.2.3 Label Editing - Set the Value of an Attribute to be Fixed / Static

Recall that in a schema you may set a fixed / static value for an attribute by inserting “fixed=<value>” in the XML element. The modified entry would look like this:

```
<xsd:element name="product_class" type="pds:product_class"
         fixed="Product_Table_Character" minOccurs="1" maxOccurs="1">
</xsd:element>
```

You have the same control over setting a fixed / static value in the label template. Use either your favorite text editor or XML aware editor to edit the label template and simply enter the value between the XML tags.

```
<product_class>Product_Table_Character</product_class>
```

7.2.4 Label Editing - Specify a Value from an Enumerated List of Values

Specifying / restricting a value, from an enumerated list of values, is very similar to setting a value to be fixed / static. The difference being that the value must correspond to one of the values in the enumerated list.

Recall that in a schema you may set a fixed / static value for an attribute by inserting "fixed=<value>" in the XML element. The modified entry would look like this:

```
<xsd:element name="unit" type="pds:unit" fixed="byte" minOccurs="0"
      maxOccurs="1">
</xsd:element>
```

Keep in mind that by setting a fixed / static value in the schema, every instance of the attribute will have that fixed value. This may not be the best approach if the attribute is repeated. This would be the case for <unit> appearing in the <Field_Character> class where there are N fields in the table and where the unit is not always 'Angstrom' in every field.

In this case, you actually have more control over setting a fixed / static value (from a list of enumerated values) in the label template. Each instance can be set independently. Use either your favorite text editor or XML aware editor to edit the label template and simply enter the value between the XML tags.

```
<Field_Character>
    <name>ELE_DEN</name>
    <field_number>2</field_number>
    <field_location>25</field_location>
    <data_type>ASCII_Real</data_type>
    <field_length>10</field_length>
    <unit>count/cm**3</unit>
    <description>
        Column contains total electron moment density in counts/cm^3.
    </description>
    <Special_Constants>
       <missing_constant>-9.99e+10</missing_constant>
    </Special_Constants>
</Field_Character>
<Field_Character>
    <name>ELE_TEMP</name>
    <field_number>3</field_number>
    <field_location>35</field_location>
    <data_type>ASCII_Real</data_type>
    <field_length>10</field_length>
    <unit>electronvolt</unit>
    <description>
        Column contains total electron moment temperature in units of
        electron volts.
    </description>
    <Special_Constants>
       <missing_constant>-9.99e+10</missing_constant>
    </Special_Constants>
</Field_Character>
```

Note that in all cases, the value or values that are specified for that particular attribute must correspond to one of the values in the enumerated set specified in the schema. It is possible that a further restriced set of values has been specified in the schematron.

7.3 XML Preamble

This section addresses how the XML preamble of an XML document is created.

```
<?xml version="1.0" encoding="UTF-8"?>
<Product_Collection xmlns="http://pds.nasa.gov/pds4/pds/v08"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v08
file:/D:/schemas/PDS4_PDS_0800k.xsd">
```

7.3.1 XML Preamble – XML Declaration

The first line is known as the XML Declaration. The XML declaration must be the first line and is static. When you create your label template, the XML declaration is auto-populated by the XML-aware editor.

```
<?xml version="1.0" encoding="UTF-8">
```

Information about this line can be found at:

http://www.w3.org/TR/REC-xml/#sec-rmd

7.3.2 XML Preamble – Product Type

The second line specifes the type of PDS product. There are two components to this line. The first leftmost component specifies one of the approved PDS product types (e.g., Product_Document, Product_Collection, etc). The second component references the URI of the component. The URI of the component is derived from the schema that references the PDS product. In our example above:

- product type: Product_Collection
- URI: xmlns="http://pds.nasa.gov/pds4/pds/v08"

Since the Product_Collection is defined in the master-schema, the URI is derived from the master-schema:

```
<?xml version="1.0" encoding="UTF-8"?>
  <!-- PDS4 XML/Schema for PDS4_0.8.0.0.k  Tue Apr 03 15:28:46 PDT 2012 -->
  <!-- Generated from the PDS4 Information Model V0.8.0.0.k -->
  <!-- *** This PDS4 product schema is a preliminary deliverable. *** -->
```

```
<!-- *** It is being made available for review and testing. *** -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://pds.nasa.gov/pds4/pds/v08"
  targetNamespace="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  version="0.8.0.0">
```

Notice in the above snippet that the "xmlns:pds" namespace is defined as a URI:

```
xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
```

This URI is carried over to the XML preamble of our example Product_Collection XML document.

```
<Product_Collection xmlns="http://pds.nasa.gov/pds4/pds/v08"
```

7.3.3 XML Preamble – Schema Instance

The third line of the XML preamble warrants that the document conforms to the underlying schema (in the XML schema instance namespace).

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

When you create your label template, the XML declaration is auto-populated by the XML-aware editor.

7.3.4 XML Preamble – Schema Location

The fourth line specifies the location of the XML schema that is referenced by the XML document. Note that there are variants as to how the schema location is specified. We will address the types of specifications.

In our example, there are two components to this line. The first leftmost component is the URI (that we have come to know and love) of the schema referenced by the Product_Collection XML document. The second rightmost component is, in this case, the physical location / path to the schema (e.g., D:/schemas/PDS4_PDS_0800k.xsd).

```
xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v08
file:/D:/schemas/PDS4_PDS_0800k.xsd">
```

When you create your label template, the XML declaration is auto-populated by the XML-aware editor. Both the URI and the location /path are derived from the schema. However, you might have to change the location / path as the label template and schema progress throught the development stage.

7.3.4.1 Schema Location – Physical Location

Here are alternate methods for specifying the physical location / path of the schema.

- Schema and label template are co-located:

```
xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v08
file:/PDS4_PDS_0800k.xsd">
```

- Schema is located in a parent directory to label template:

```
xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v08
file:./PDS4_PDS_0800k.xsd">
```

7.3.4.2 Schema Location – Online Accessible Location

Another method for referencing the location of the schema is through a reference to a URL that is online accessible.

- Schema is online accessible:

```
xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v08
http://pds.nasa.gov/schemas/PDS4_PDS_0800k.xsd">
```

7.3.4.3 Schema Location – Using XML Catalog

Yet another method for referencing the location of the schema is by using an XML catalog.

```
xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v08 junk.xsd">
```

For additional information on using an XML catalog to reference the schema, consult the appendix on XML Catalog Reference.

7.4 XML Preamble with Local Dictionary

This section addresses how the XML preamble of an XML document, having a local dictionary, is created.   Most of your Observational products will presumably contain references to one or more local dictionaries. Each dictionary will reside in a namespace.

Recall from our previous Product_Collection XML preamble that the only namespace that was specified was the “pds” namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<Product_Collection xmlns="http://pds.nasa.gov/pds4/pds/v08"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v08
file:/D:/schemas/PDS4_PDS_0800k.xsd">
```

The pramble of XML documents, that reference local dictionaries, require each namespace to be identified. Our next example includes a reference to a single local dictionary in the "dph" namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:dph="http://pds.nasa.gov/schema/pds4/dph/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v08
http://pds.jpl.nasa.gov/pds4/schemas/PDS4_PDS_0800k.xsd">
```

From the above example, you can see that the "dph" namespace has the following URI:

```
xmlns:dph="http://pds.nasa.gov/schema/pds4/dph/v01"
```

If you have been following the association between the schema and the referenced URI, you will not be surprised that the "dph" local dictionary specifies the URI that is to be used when referencing the "dph" schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://pds.nasa.gov/schema/pds4/dph/v01"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:dph="http://pds.nasa.gov/schema/pds4/dph/v01"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified" version="0.8.0.0.k">
```

Had the XML document referenced multiple local dictionaries, you simply repeat the reference to each namespace identified by each local dictionary.

```
<?xml version="1.0" encoding="UTF-8"?>
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:dph="http://pds.nasa.gov/schema/pds4/dph/v01"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v08
http://pds.jpl.nasa.gov/pds4/schemas/PDS4_PDS_0800k.xsd">
```

In the above example, both the "dph" and the "phxmd" local dictionaries are referenced.

7.5 Next Steps

In any case, you will want to have a fully functional PDS4 compliant label template in place before proceeding onto the next step of integrating the pieces into the pipeline production of the various products in your archive.

7.6 Example XML Preambles

For additional information regarding the XML preamble and they are used within a Product Label, consult your discripline node.

For examples, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

PART IV. COLLECTIONS, BUNDLES, DELIVERY PACKAGES

8.0 COLLECTIONS

The next higher level in the organizational hierarchy of an archive is the Collection — an inventory of member products and an accompanying label (including an identifier). The inventory and label are known as a Collection Product. Products of similar type and content are grouped into a collection. Observational data, for example, will be gathered into an observational data collection; documents into a document collection; and supplementary data into a supplementary collection.

In consultation with your DN, for each product type, determine how best to group the products, that share common characteristics, into appropriate collections.

To define a collection, the archivist creates a collection inventory, a specific type of character table, that lists all the product label files (XML) and their logical identifiers (LIDs) that are part of the collection. The archivist then creates a collection label that describes the collection inventory.

A collection label contains a logical identifier (LID) that uniquely identifies the collection and provides the links between products that share common characteristics.

The products listed in the collection inventory must be physically located either in the same directory as the collection label, or in one or more subdirectories to it.

For information on how to populate the attributes and classes in a collection, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

8.1 Members of a Collection

The Members of a collection are designated as being either primary or secondary.

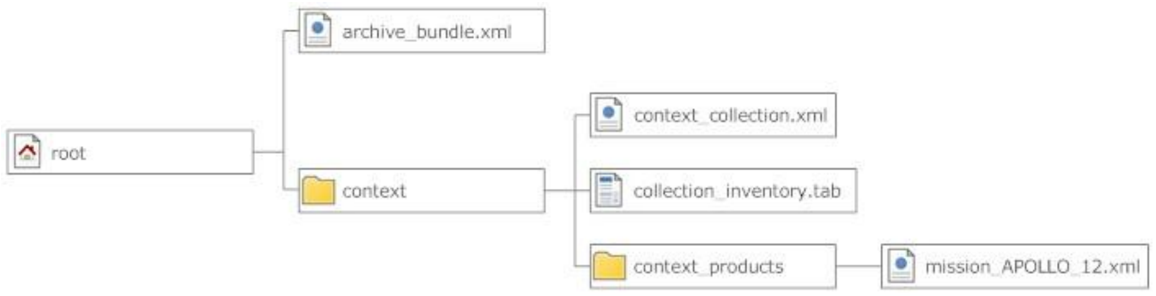
- Every product must be a primary member of one and only one collection.
    - That collection is the one in which the product is registered in the PDS repository.
- Products already registered in the PDS repository may be secondary members of other collections.
    - For example a collection of mosaic products might include the source products for each mosaic as secondary members of the collection.
- Primary members must be identified in the collection inventory using LIDVIDs.

- Secondary members may be identified in the collection inventory using either LIDs or LIDVIDs based on which is more appropriate for that collection and product.

8.2 Collection Inventory

Each collection product consists of two files, a collection XML label and a collection inventory. Depending upon the type of collection, the collection inventory, a listing of the products that are the members of the collection, is either a one or two column character table where each row of the table describes one of the member products of the collection.

The first column of the table specifies the LID or LIDVID of the product. The second column of the table is the file_specification_name of the product. The file_specification_name defines the name and location of the product relative to the location of the collection product.



In the above diagram for the mission_APOLLO_12 context product:

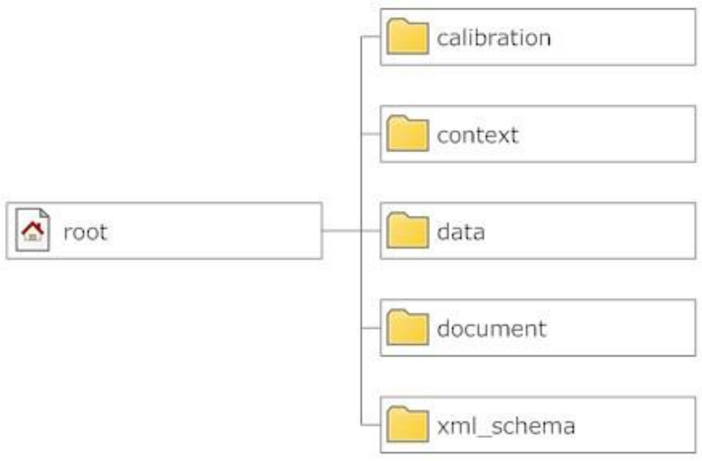
1. The LIDVID would be the value of the <logical_identifer> in the mission_APOLLO_12.xml file.
2. The file_specification_name would be "context_products/mission_APPOLO_12.xml".

Note that directory paths are always specified in UNIX system format (i.e., the forward slash ("/") is used to denote directories and to delineate subdirectories).

8.3 Generating and Populating a Collection Label

Using an XML aware editor, the archivist generates a label template using the master-schema and selecting the Product_Collection class.

You will create a collection product for each type of product in the archive. For example, if your archive has the following directory structure:



You would create a collection / inventory for the products contained in each of the above directories.

8.4 Collection Product Label – Block by Block

8.4.1 Identification Area

The Identification Area block in the template looks something like this:

```
<Identification_Area>
    <logical_identifier>logical_identifier0</logical_identifier>
    <version_id>version_id0</version_id>
    <title>title0</title>
    <information_model_version>
      information_model_version0
    </information_model_version>
    <product_class>product_class0</product_class>
    <Alias_List>
        <Alias>
            <alternate_id>alternate_id0</alternate_id>
            <alternate_title>alternate_title0</alternate_title>
            <comment>comment0</comment>
        </Alias>
    </Alias_List>
    <Citation_Information>
        <author_list>author_list0</author_list>
        <editor_list>editor_list0</editor_list>
        <publication_year>publicatio</publication_year>
        <keywords>keywords0</keywords>
        <description>description0</description>
    </Citation_Information>
    <Modification_History>
        <Modification_Detail>
            <modification_date>0000</modification_date>
            <version_id>version_id1</version_id>
            <description>description1</description>
        </Modification_Detail>
    </Modification_History>
</Identification_Area>
```

The Identification Area of the collection is identical to that of the Basic Product (i.e., there are no differences in populating the Identification_Area of either the collection or the Basic Product. See Section 3.2 in the Standards Reference [2] for additional information on populating the Identification_Area.

8.4.2 Subject Area

Populating entries in the Subject_Area of the collection is identical to that of the Basic Product. See Section 3.2.1 in the Standards Reference [2] for additional information on populating the Observation_Area.

8.4.3 Reference_List Area

Populating entries in the Reference_List area of the collection is identical to that of the Basic Product. See Section 3.x in the Standards Reference [2] for additional information.

8.4.4 Collection Area

The Collection area contains a single attribute – collection_type. For each collection (e.g., calibration, context, etc), you will indicate the type of collection being described by setting the value of the collection_type attribute in the Collection class to one of the following values:

- Browse
- Calibration
- Context
- Data
- Document
- Geometry
- Miscellaneous
- SPICE
- Schema

8.4.5 File Area

The File Area of a collection product is used to reference primary and secondary products. The File_Area_Inventory class has specific purposes with respect to referencing primary and / or secondary products.

If you are still uncertain as to whether or not the products to be inventoried are primary or secondary, consult your PDS discipline node.

We are going to use the example of a collection having both primary and secondary products. If your collection only has either primary or secondary products, then tailor the following steps accordingly.

In our example, your collection will consist of three files:

- collection.xml
- collection_inventory_primary.tab
- collection_inventory_secondary.tab

The collection.xml XML document will reference both the collection_inventory_primary.tab and the collection_inventory_secondary.tab files. The collection_inventory_primary.tab will contain the inventory of the primary products and the collection_inventory_secondary.tab will contain the inventory of the secondary products.

8.4.5.1 File Area Inventory (Primary and Secondary)

Since our example will have both primary and secondary products, we will create two instances of the File_Area_Inventory class; one class for describing the primary members and a second class for describing the secondary members.

The two instances of File_Area_Inventory are in essence identical and are populated near-identically. The principle difference is that the two areas uniquely identify the type of product being inventoried --- primary or secondary.

The principle difference is how the reference_type attribute, in the Inventory class, is set to indicate an inventory of either primary or secondary products:

- has_member_LIDVID_Primary – used to inventory primary products
- has_member_LIDVID_Secondary – used to inventory secondary products.

```
<File_Area_Inventory>
    <File>
        <file_name>collection_browse_inventory_20110823.tab</file_name>
        <local_identifier>collection_browse_inventory</local_identifier>
        <creation_date_time>2011-08-23T11:53:24.5749Z</creation_date_time>
        <file_size>513</file_size>
        <records>1</records>
        <md5_checksum>899fba057920491a08c7d517bd65e717</md5_checksum>
    </File>
    <Inventory>
        <local_identifier>dph_example_collection</local_identifier>
        <offset unit=”byte”>0</offset>
```

```
            <records>13</records>
            <encoding_type>CHARACTER</encoding_type>
            <reference_type>has_member_LIDVID_Primary</reference_type>
              .
              .
              .
        </Inventory>
    </File_Area_Inventory>
```

Both instances of the File_Area_Inventory contain a File area and an Inventory area. The File Area of the collection product XML label document is used to reference the data file(s) (i.e., the string of bits) being described by our collection label. In our example, there will be two File areas – one for each of the inventory files. The File area for a collection product is populated identically to that of a Basic Product. Refer to Section 3.5 in the Standards Reference [2] for additional information.

The inventory table has special properties based on the type of products being inventoried.

Type of Product (reference_type)	**Inventory Table: column #1**	**Inventory Table: column #2**
inventory_has_LIDVID_Primary	(r) LIDVID	(r) ASCII_File_Specification_Name
inventory_has_LIDVID_Secondary	(r) LIDVID	(o) ASCII_File_Specification_Name
inventory_has_LID_Secondary	(r) LID	(o) ASCII_File_Specification_Name

(r) = required
(o) = optional

The Inventory areas provide the information that describes a simple one or two column character table where each row of the table describes one of the products in the collection. The first column of the table is the LID or LIDVID of the product. The second column of the table is the file_specification_name of the product (if supplied). The file_specification_name defines the name and location of the product relative to the location of the collection product.

The following label snippet is an example of how a File_Area_Inventory area is populated with metadata to describe an inventory of primary products (i.e., a two column character table). The table describes 10 records / rows of data, where each row is 513 bytes; each column contains 255 characters, and each row is terminated with <CR><LF>.

```
    <File_Area_Inventory>
        <File>
            <file_name>inventory_context_collection_20110823.tab</file_name>
            <local_identifier>inventory_context_collection</local_identifier>
            <creation_date_time>
               2011-08-23T11:53:24.5749Z
            </creation_date_time>
            <file_size>2052</file_size>
            <records>4</records>
            <md5_checksum>899fba057920491a08c7d517bd65e717</md5_checksum>
        </File>
        <Inventory>
            <local_identifier>dph_example_collection</local_identifier>
            <offset unit=”byte”>0</offset>
            <records>10</records>
            <encoding_type>CHARACTER</encoding_type>
            <reference_type>has_member_LIDVID_Primary</reference_type>
            <Record_Character>
               <record_length>513</record_length>
               <fields>2</fields>
               <Field_Character>
                  <name>LIDVID</name>
                  <field_number>1</field_number>
                  <field_location>1</field_location>
                  <data_type>ASCII_LIDVID</data_type>
                  <field_length>255</field_length>
                  <field_format>urn:nasa:pds:xxxx::M.n</field_format>
                  <description>
                      This columns specifies the LIDVID of the files that
                      comprise the collection.
                  </description>
               </Field_Character>

               <Field_Character>
                <name>file_specification_name</name>
                <field_number>2</field_number>
                <field_location>257</field_location>
                <data_type>ASCII_File_Specification_Name</data_type>
                <field_length>255</field_length>
                <description>
                    This columns specifies the location of the files that
                    comprise the collection where the location is specified
                    relative to the location of the collection.
                </description>
               </Field_Character>
            </Record_Character>
        </Inventory>
    </File_Area_Inventory>
```

In the above label snippet, both columns have been set to 255 characters which is the maximum number of characters that a LIDVID and a file_specification_name can have. The two columns of data are separated by a <comma> character. The two columns of data in the inventory file would be padded so that the 2nd column of data starts at byte position 257 and the <CR><LF> characters occupy positions 512 and 513.

```
         1         2               250       260               510       520
12345678901234567890 . . . 12345678901234567890 . . . 12345678901234567890
urn:nasa:pds:DPH:root::1.0                ,context/readme.txt    CL
```

An alternative, to padding the columns to the maximum length, would be to adjust the position of the 2nd column to start at a byte position that is greater than the maximum length of the values in the 1st column. For example, if the longest LIDVID is 50 characters in length, then the 2nd column could start at byte position 52.

We can easily adjust the above label snippet to describe an inventory of secondary products (i.e., a one column character table). Simply adjust the reference_type and fields attributes and remove the second instance of Field_Character.

```
<File_Area_Inventory>
    <File>
        <file_name>inventory_context_collection_20110823.tab</file_name>
        <local_identifier>inventory_context_collection</local_identifier>
        <creation_date_time>
           2011-08-23T11:53:24.5749Z
        </creation_date_time>
        <file_size>2052</file_size>
        <records>4</records>
        <md5_checksum>899fba057920491a08c7d517bd65e717</md5_checksum>
    </File>
    <Inventory>
        <local_identifier>dph_example_collection</local_identifier>
        <offset unit="byte">0</offset>
        <records>10</records>
        <encoding_type>CHARACTER</encoding_type>
        <reference_type>has_member_LIDVID_Secondary</reference_type>
        <Record_Character>
           <record_length>513</record_length>
           <fields>1</fields>
           <Field_Character>
              <name>LIDVID</name>
              <field_number>1</field_number>
              <field_location>1</field_location>
              <data_type>ASCII_LIDVID</data_type>
              <field_length>255</field_length>
              <field_format>urn:nasa:pds:xxxx::M.n</field_format>
              <description>
                  This columns specifies the LIDVID of the files that
                  comprise the collection.
              </description>
           </Field_Character>
        </Record_Character>
    </Inventory>
</File_Area_Inventory>
```

8.5 XML Schema Collection

Probably one of the last collections that you will create will be the XML schema collection. This collection consists of an inventory of all of the schemas that were used in creating the products in your archive.

The collection label for the xml_schemas is virtually identical to the other collection labels.

8.6 Example Collection Product Label

For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of a Collection label, consult the PDS4 Data Dictionary. For information on how to populate the attributes and classes in a collection, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

9.0 BUNDLES

The highest level product is referred to as a Bundle — an inventory of member products and an accompanying label (including an identifier). Like collections, bundles consist of a list of references to products; however in this case, the referenced products are collections. A bundle identifies all of the collections in the archive; where the collection; in turn, identifies all of the basic products necessary to perform useful science analysis on the data contained therein. The inventory and label are known as a Bundle Product. Unlike collection products, the bundle inventory is a table contained in the XML label file describing it, not in a separate table file.

To define a bundle, the archivist creates a Product_Bundle label. A bundle label contains a logical identifier (LID) that uniquely identifies the bundle and provides the links between the collection products in the archive.

For information on how to populate the attributes and classes in a bundle, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

9.1 Generating and Populating a Bundle Label

Using an XML aware editor, the archivist generates the Product_Bundle label.

You will create a single bundle product for the archive with references to the member collections in the archive.

9.2 Identification Area

Populating entries in the Identification_Area of the bundle is identical to that of the Collection and the Basic Product. See Section 3.2 in the Standards Reference [2] for additional information on populating the Identification_Area.

9.3 Subject Area

Populating entries in the Subject_Area of the bundle is identical to that of the Collection and the Basic Product. See Section 3.4 in the Standards Reference [2] for additional information.

9.4 Reference_List Area

Populating entries in the Reference_Area of the bundle is identical to that of the Collection and the Basic Product. See Section 3.x in the Standards Reference [2] for additional information.

9.5 Bundle Area

The Bundle area contains a two attributes – bundle_type and description. This is where you, the data provider, will want to adequately describe the bundle and indicate the type of bundle that you are producing. Since you are creating a bundle that describes an archive, you will set bundle_type to "Archive". A non-archive bundle has a bundle_type of "Supplemental".

```
<Bundle>
    <description>
        Some really good description of the archive goes here
    </description>
    <bundle_type>Archive</bundle_type>
</Bundle>
```

9.6 File_Area_Text

The File_Area_Text of a bundle product XML label is used to reference the 'readme.txt' file (i.e., the string of bits) being described by the label.

The File_Area_Text class in the template looks something like this:

```
<File_Area_Text>
     <File>
         <file_name>file_name0</file_name>
         <local_identifier>local_identifier2</local_identifier>
         <creation_date_time>0000-01-01T00:00</creation_date_time>
         <file_size>0</file_size>
         <records>50</records>
         <maximum_record_bytes>50</maximum_record_bytes>
         <md5_checksum>00000000000000000000000000000000</md5_checksum>
         <comment>comment6</comment>
     </File>
     <Stream_Text>
         <name>name6</name>
         <local_identifier>local_identifier3</local_identifier>
         <offset unit="byte">50</offset>
         <external_standard_id>external_standard_id0</external_standard_id>
         <encoding_type>encoding_type0</encoding_type>
         <description>description11</description>
     </Stream_Text>
 </File_Area_Text>
```

Populating entries in the File_Area_Text of the XML label is remarkably straightforward. The file block consists of a number of optional and required attributes that describe the data file.

- The file name attribute, a required attribute, provides the name of the file being described.
- The local identifier attribute, an optional attribute, provides the name of the local object being described; the value must be unique within the XML label.
- The creation date time attribute, an optional attribute, provides the date/time when the file was created, either in YMD or DOY format.
- The file size attribute, an optional attribute, provides the size (in bytes) of the file.
- The records attribute, am optional attribute, provides the number of records in the file.
- The max record bytes attribute, an optional attribute, provides the size (in bytes) of the maximum record within the file. For example, in a character table having a fixed-length 78 byte record structure, the max_record_bytes is set to 78 (which is inclusive of the record delimiters).
- The md5 checksum attribute, an optional attribute, provides the md5 checksum of the file.
- The comment attribute, an optional attribute, provides a brief description of the comment.

The next block, the Stream_Text class, contains the reference to the 'readme.txt' product component being described by the XML label.

```
<Stream_Text>
    <local_identifier>local_identifier1</local_identifier>
    <comment>comment1</comment>
    <encoding_type>Character</encoding_type>
    <external_standard_id>standard_id0</external_standard_id>
    <offset unit="byte">0</offset>
</Stream_Text>
```

1. As the 'readme.txt' file is always ASCII, the encoding type is set to 'Character'.
2. The offset is set to indicate the starting location (in bytes) of the 'readme.txt' object. Populating the offset attribute for the bundle product is very simple. As there is only a single product being referenced, the offset is always set to '0' bytes.

Note that the 'readme.txt' file referenced by the bundle product label must be co-located with the Product_Bundle product label.

Additional information on the File_Area_Text and how it is used within the archive can be found in the Standards Reference [2].

9.7 Bundle Member Entry

The Members of a bundle are referenced using the Bundle_Member_Entry area. The Bundle_Member_Entry area is repeated for each collection product.

```
<Bundle_Member_Entry>
    <lid_reference>urn:nasa:pds:A</lid_reference>
    <file_specification_name>file_specification_name0</file_specification_name>
    <member_status>member_status0</member_status>
    <reference_type>reference_type4</reference_type>
</Bundle_Member_Entry>
```

In the Bundle_Member_Entry class:

1. You should use / specify either the lid_reference or the lidvid_reference attribute. You normally will only specify one or the other. But, you can specify both the lid_reference and the lidvid_reference as long as both reference the identical product.
2. The value for the reference_type is dependent up on the type of collection being referenced:

• Browse	- bundle_has_browse_collection
• Calibration	- bundle_has_calibration_collection
• Context	- bundle_has_context_collection
• Data	- bundle_has_data_collection
• Document	- bundle_has_document_collection
• Geometry	- bundle_has_geometry_collection
• Miscellaneous	- bundle_has_member_collection
• SPICE	- bundle_has_spice_kernel_collection
• Schema	- bundle_has_schema_collection

Note that the collection products referenced by a bundle label must either be co-located with the bundle label, or in a subdirectory of the directory containing the bundle label.

The Bundle_Member_Entry area is used to specify the location of the collection product. The value specified in the file_specification_name attribute is always relative to the location of the bundle label. If the bundle label and the collection label are co-located, then the value is simply the name of the XML label file:

<file_specification_name>filename.tab</file_specification_name>

If the collection XML label was located in the “document” subdirectory, then the value would be:

<file_specification_name>document/collection.xml</file_specification_name>

Note that directory paths are always specified in UNIX system format (i.e., the forward slash (“/”) is used to denote directories and to delineate subdirectories).

9.8 Example Bundle Product Label

For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of a Bundle label, consult the PDS4 Data Dictionary. For information on how to populate the attributes and classes in a bundle, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

10.0 DELIVERIES

Transfers of products from point to point within and external to the PDS can be accomplished using a variety of mechanisms from electronic transfer by email attachment of a handful of files to the transfer of 2TB of files using external hard drives. The actual transfer mechanism should be coordinated with the node with which you are working.

Any transfer, to, from, or within PDS has two parts, the "Package" containing the material being transferred and the "Delivery Manifest" describing those materials.

10.1 The Package

Transfer packages must be approved by PDS and include options such as ZIP, gzip, tar, physical media such as thumb drives, external hard drives, etc. There is no requirement to use any of these for the transfer of a handful of files, nor is there a requirement to use software such as zip when the entire transfer is being made with a dedicated external hard drive.

The data provider and the receiving node determine the best 'packaging' option for the delivery.

Except for the case of the delivery of a handful of files, either as test samples or as replacements for previously delivered files, the files within the delivery package must be organized in a PDS compliant directory structure (see the PDS Standards Reference, Section x.x for details.

10.2 The Delivery Manifest

The delivery manifest is a file provided to PDS external to the package. It is a checksum manifest for each file in the package. Please note if the delivery package is a zip file, tar file, etc., the manifest must contain entries for every file in the package once that package is unpacked. Currently PDS uses MD5 checksums, so the delivery manifest is just an MD5 checksum manifest. Since it is external to the package, and hence external to the archive, there is no requirement to provide a PDS4 XML label for a delivery manifest.

10.3 Generating and Populating a Delivery_Manifest Label

Using an XML aware editor, the archivist generates the Product_Delivery_Manifest label.

10.4 Identification Area

Populating entries in the Identification_Area of the delivery_manifest is identical to that of the Bundle, Collection, and the Basic Product. See Section 3.2 in the Standards Reference [2] for additional information on populating the Identification_Area.

10.5 Reference_List Area

Populating entries in the Reference_Area of the delivery_manifest is identical to that of the Bundle, Collection, and the Basic Product. See Section 3.x in the Standards Reference [2] for additional information.

10.6 File_Area_Delivery_Manifest

The File_Area_Delivery_Manifest class of a delivery_manifest product XML label is used to reference the 'manifest.txt' file (i.e., the string of bits) being described by the label.

The File_Area_ Delivery_Manifest class in the template looks something like this:

```
<File_Area_Delivery_Manifest>
     <File>
         <file_name>file_name0</file_name>
         <local_identifier>local_identifier0</local_identifier>
         <creation_date_time>0000-01-01T00:00</creation_date_time>
         <file_size>0</file_size>
         <records>50</records>
         <maximum_record_bytes>50</maximum_record_bytes>
         <md5_checksum>00000000000000000000000000000000</md5_checksum>
         <comment>comment2</comment>
     </File>
     <Delivery_Manifest>
         <local_identifier>local_identifier1</local_identifier>
         <offset unit="byte">50</offset>
         <records>50</records>
         <encoding_type>encoding_type0</encoding_type>
         <description>description3</description>
         <reference_type>reference_type1</reference_type>
        <Record_Character>
             <record_length unit="byte">50</record_length>
             <fields>50</fields>
             <Field_Character>
                 <name>name6</name>
                 <field_number>50</field_number>
                 <field_location unit="byte">50</field_location>
                 <data_type>data_type0</data_type>
                 <field_length unit="byte">50</field_length>
                 <field_format>field_format0</field_format>
                 <unit>unit1</unit>
                 <scaling_factor>0</scaling_factor>
                 <value_offset>0</value_offset>
                 <description>description11</description>
                 <Special_Constants>
                 </Special_Constants>
                 <Field_Statistics>
                 </Field_Statistics>
             </Field_Character>
```

```
            </Record_Character>
        </Delivery_Manifest>
    </File_Area_Delivery_Manifest>
```

Populating entries in the File_Area_Delivery_Manifest label is remarkably straightforward. The file block consists of a number of optional and required attributes that describe the data file.

- The file name attribute, a required attribute, provides the name of the file being described.
- The local identifier attribute, an optional attribute, provides the name of the local object being described; the value must be unique within the XML label.
- The creation date time attribute, an optional attribute, provides the date/time when the file was created, either in YMD or DOY format.
- The file size attribute, an optional attribute, provides the size (in bytes) of the file.
- The records attribute, am optional attribute, provides the number of records in the file.
- The max record bytes attribute, an optional attribute, provides the size (in bytes) of the maximum record within the file. For example, in a character table having a fixed-length 78 byte record structure, the max_record_bytes is set to 78 (which is inclusive of the record delimiters).
- The md5 checksum attribute, an optional attribute, provides the md5 checksum of the file.
- The comment attribute, an optional attribute, provides a brief description of the comment.

The next block, the Delivery_Manifest class, contains the reference to the ‘manifest.txt’ product component being described by the XML label.

```
        <Delivery_Manifest>
            <local_identifier>local_identifier1</local_identifier>
            <offset unit="byte">50</offset>
            <records>50</records>
            <encoding_type>CHARACTER</encoding_type>
            <description>description goes here</description>
            <record_delimiter></record_delimiter>
           <Record_Character>
                <record_length unit="byte">274</record_length>
                <fields>2</fields>
                <Field_Character>
                    <name>ASCII_MD5_Checksum</name>
                    <field_number>1</field_number>
                    <field_location unit="byte">1</field_location>
                    <data_type>ASCII_MD5_Checksum</data_type>
                    <field_length unit="byte">15</field_length>
                    <field_format>ASCII_Short_String_Collapsed</field_format>
                    <description>
                      First column is ASCII_MD5_Checksum of the
                      product being described
                    </description>
                </Field_Character>
                <Field_Character>
                    <name>ASCII_File_Specification_Name</name>
                    <field_number>2</field_number>
                    <field_location unit="byte">17</field_location>
                    <data_type>ASCII_Short_String_Collapsed</data_type>
                    <field_length unit="byte">255</field_length>
                    <field_format>ASCII_Short_String_Collapsed</field_format>
                    <description>
                      Second column is file_specification_name of the
                      product being described
                    </description>
                </Field_Character>
            </Record_Character>
        </Delivery_Manifest>
```

1. As the ‘manifest.txt’ file is always ASCII, the encoding type is set to ‘Character’.
2. The offset is set to indicate the starting location (in bytes) of the manifest. Populating the offset attribute for the delivery_manfiest product is very simple. As there is only a single product being referenced, the offset is always set to ‘0’ bytes.
3. The records attributes indicates the number of records / products being described / delivered.
4. The encoding_type attribute is set to ‘Character’.
5. The description attribute provides a terse description of the delivery.
6. The record_delimiter is set to indicate the type of delimiter (e.g., ‘carriage_return’, ‘carriage_return line_feed’, ‘line_feed’).
7. The record_length attribute is set to a maximum length of ‘274’ characters; assuming you will pad the second column to a length of 255 characters. The length of the first column is fixed at ‘17’ characters.
8. The fields attribute is set to ‘2’.
9. The name of the first Field_Character is set to ‘ASCII_MD5_Checksum’.
10. The field_number of the first Field_Character is set to ‘1’.
11. The field_location of the first Field_Character is set to ‘1’.
12. The data_type of the first Field_Character is set to ‘ASCII_MD5_Checksum’.
13. The field_length of the first Field_Character is set to ‘15’ which is the length of MD5 checksums.
14. The field_format of the first Field_Character is set to ‘ASCII_Short_String_Collapsed’.
15. The description attribute provides a terse description of the MD5 column.
16. The name of the second Field_Character is set to ‘ASCII_File_Specification_Name’.
17. The field_number of the second Field_Character is set to ‘2’.
18. The field_location of the second Field_Character is set to ‘17’.
19. The data_type of the second Field_Character is set to ‘ASCII_Short_String_Collapsed’.
20. The field_length of the second Field_Character is set to a maximum length of ‘255’ which is the maximum length of a file-specification-name. The value can represent the maximum length of the value specified.
21. The field_format of the second Field_Character is set to ‘ASCII_Short_String_Collapsed’.

22. The description attribute provides a terse description of the MD5 column.

Note that the ‘manifest.txt’ file referenced by the delivery_manifest product label must be co-located with the Product_Delivery_Manifest product label.

Additional information on the Product_Delivery_Manifest product and how it is used within the archive can be found in the Standards Reference[2].

10.7 Delivery of Accumulating Archives

The packaging and manifest constraints are exactly the same for incremental deliveries. However, it is neither required, nor in general desirable, for a provider to redeliver files which have not changed.

Consider a single collection with new files submitted quarterly to the PDS using an external hard drive. Each incremental delivery will have an updated collection product in the root directory, a sufficient portion of the collection directory structure for each of the files in the delivery to be placed appropriately, and all of the new files. It will also include a delivery manifest for that delivery; however the manifest must be provided to the PDS separately – not on the same external hard drive.

See the PDS Standards Reference, Section x.x, for rules regarding Version ID incrementing for incremental deliveries.

10.8 Example Delivery Product Label

For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of a basic product label, consult the PDS4 Data Dictionary.

PART V. LOCAL DICTIONARIES

11.0 LOCAL DATA DICTIONARY

A Data Dictionary serves several purposes. First, the dictionary serves as a reference manual to users of the PDS (and other planetary data systems) to define the attributes and classes that are used to describe planetary data and meta-data. Second, the dictionary serves as a reference for data producers (and others) to aid in the design and understanding of data descriptions. Third, the dictionary has the overall responsibility of ensuring the attributes and classes used in the data descriptions are used in a standard, consistent, predictable manner to the point where each attribute and class can be managed and used as a resource.

Conceptually, a data dictionary defines the attributes and classes which may be used in PDS4 product labels. Practically speaking, it must contain human-readable definitions as well as the syntax and semantic constraints placed on values of the attribute. For classes, it provides the explicit list of attributes constituting the class, and indicates which are required, optional, and/or repeatable. It might also indicate that one or more sub-classes are allowed (or required).

Every attribute and class that is used in any PDS label must first be defined in a data dictionary. Ultimately, all dictionaries will be integrated into the PDS4 Information Model which will "build" the PDS4 Data Dictionary document and the associated Mission and Node "dictionary" schemata.

Data dictionaries are categorized into:

- Mission specific
- Node specific

Mission specific data dictionaries are those that are comprised of attributes and classes specific to a particular mission or investigation (i.e., the Mars Express mission could create a data dictionary for the sole purpose of defining attributes and classes that describe instrumentation and science data that is particular to the Mars Express mission). Depending upon preference, the mission may elect to have a single data dictionary (that describes both instrumentation and science data) or multiple dictionaries where one dictionary is specific to instrumentation descriptors and another is specific science data descriptors.

Node specific data dictionaries are those that are comprised of attributes and classes specific to a particular PDS node. For example, the Rings node has attributes / classes that are particular to a large number of Rings products. The Rings node may identify a number of instances where it would be convenient to group Rings descriptors into one or more data dictionaries.

11.1 Building and Using Local Data Dictionaries

Local data dictionaries are categorized into:

- Mission specific
- Node specific

This section describes the inter-relationships between the data dictionary schemas, the data dictionary label(s), and the dictionary service that creates the local data dictionary schema.

The dictionary service optionally merges the attributes and classes defined in the local data dictionary with the PDS4 Information Model. This will ensure that your locally defined attributes and/or classes will be contained in the next build of the PDS4 Data Dictionary and the next build of the Generic Mission & Node schemata.

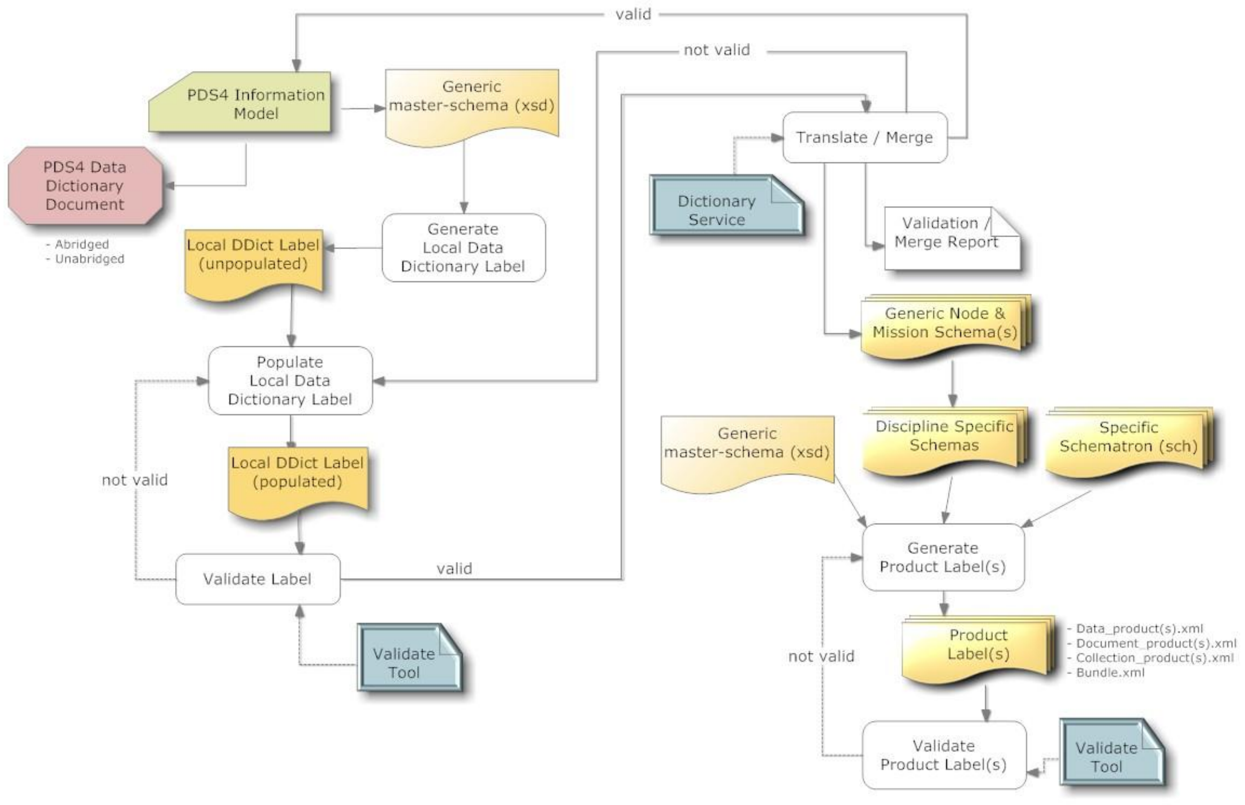

valid
not valid
PDS4 Information Model
Generic master-schema (xsd)
PDS4 Data Dictionary Document
- Abridged
- Unabridged
Local DDict Label (unpopulated)
Generate Local Data Dictionary Label
Populate Local Data Dictionary Label
Local DDict Label (populated)
not valid
Validate Label
valid
Validate Tool
Translate / Merge
Dictionary Service
Validation / Merge Report
Generic Node & Mission Schema(s)
Generic master-schema (xsd)
Discipline Specific Schemas
Specific Schematron (sch)
Generate Product Label(s)
Product Label(s)
- Data_product(s).xml
- Document_product(s).xml
- Collection_product(s).xml
- Bundle.xml
not valid
Validate Product Label(s)
Validate Tool

Figure 11-1 Local Data Dictionary Development Flow Diagram

Note that the following examples use Oxygen as the vehicle for demonstrating the local-DD process. In no way does suggest or imply a recommendation or endorsement for using Oxygen over any of the other XML-aware editors.

The steps that follow are a narrative of the major steps to creating a label-template for Ingest_DD. A more detailed description of the generic process of creating a label-template, complete with illustrations, can be found on the PDS Wiki:

https://oodt.jpl.nasa.gov/wiki/display/pdscollaboration/Root%2C+Mission%2C+and+Node+Schema+to+Label+Template

Step #1: Download the OPS version of the master-schema:

http://pds/schema/pds4/dev/generic/ops/

You will want to copy the master-schema (xsd) file to a local directory / folder.

- Open your favorite browser
- Navigate to the above url
- Select the file of choice to download
- Put your mouse-cursor over the file to be copied
  - right-click and select "Save Link-As" – this will open a dialog box to specify a directory / folder where you want to copy the file

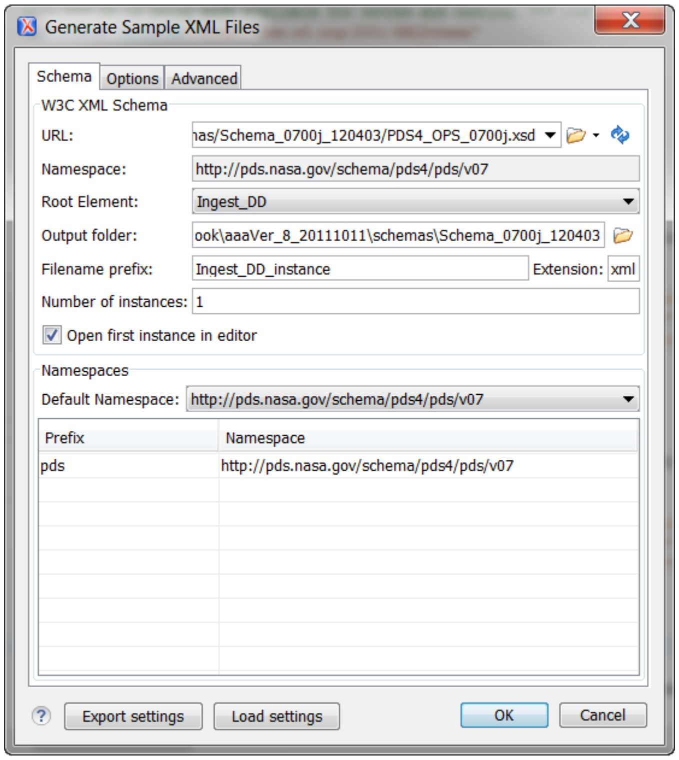
Note that the PDS version of the master-schema does not contain the imbedded reference to the class that we will want to use. Only the OPS version of the master-schema contains an embedded reference to the Ingest_DD class.

Step #2: Using the XML-aware editor of your choice, open the master-schema (xsd) file. Locate the Ingest_DD class. This is the class from which we will generate the label-template.

Step #3: Most XML-aware editors provide a capability to generate a label-template from a schema. In our case, you will wanto to generate a label-template from the reference to the Ingest_DD class.

In Oxygen:

- select: Tools / Generate Sample XML Files.



W3C XML Schema

URL: ɿas/Schema_0700j_120403/PDS4_OPS_0700j.xsd

Namespace: http://pds.nasa.gov/schema/pds4/pds/v07

Root Element: Ingest_DD

Output folder: ook\aaaVer_8_20111011\schemas\Schema_0700j_120403

Filename prefix: Ingest_DD_instance Extension: xml

Number of instances: 1

[✓] Open first instance in editor

Namespaces

Default Namespace: http://pds.nasa.gov/schema/pds4/pds/v07

Prefix	Namespace
pds	http://pds.nasa.gov/schema/pds4/pds/v07

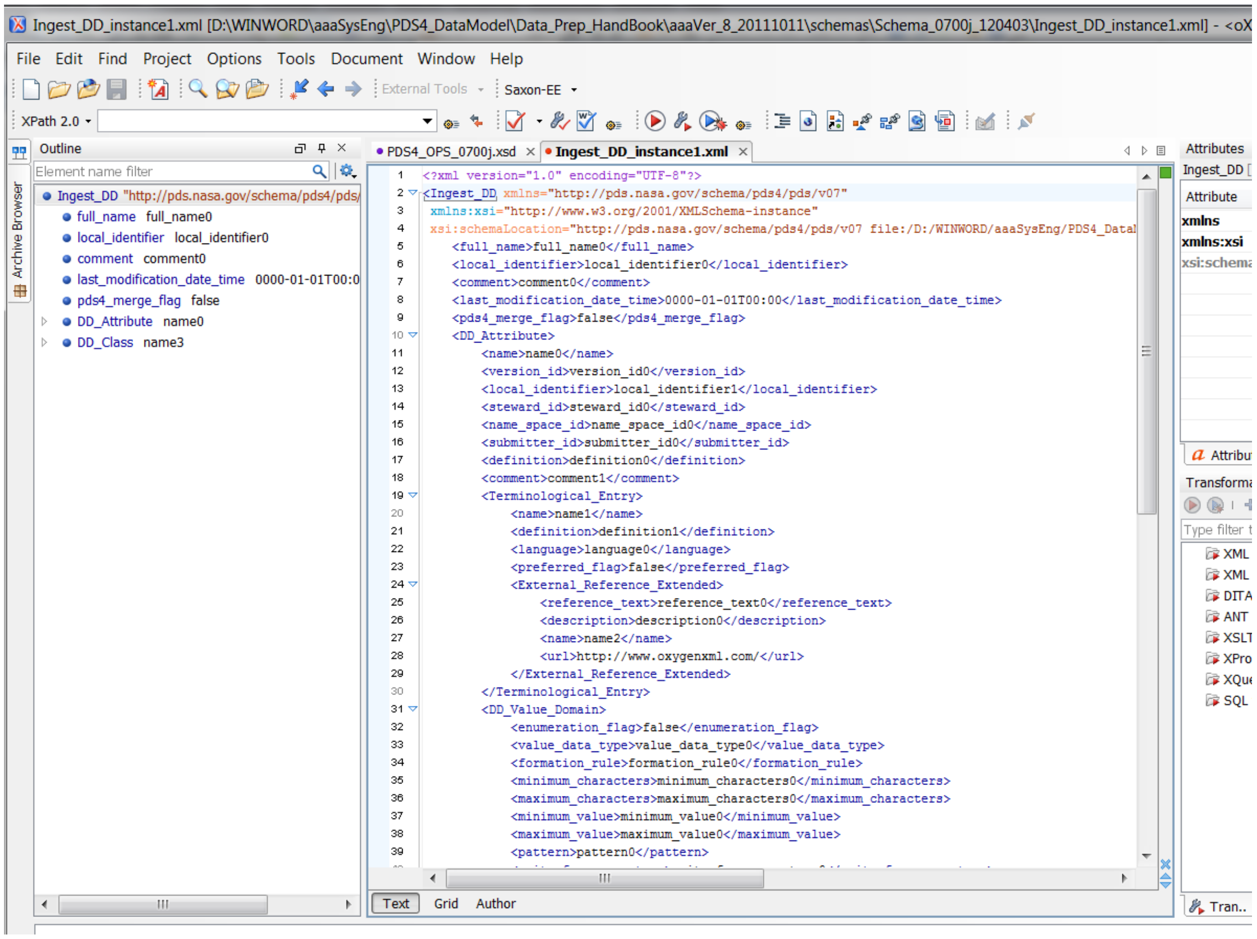
Notice that the "Schema" tab is active. This tab is used to specify the XSD from which the label template will be generated.

- In URL, you will want to specify the local copy of the master-schema file
- In Namespace, the value should be auto-populated with the value specified in the master-schema. The value should be fairly close to the value displayed above.
- In Root Element, you will want to use the drop-down to select the product for which you want to create a label template – Ingest_DD.
- In Output folder, you will want to specify the directory / folder where you want the label template to reside (when created).
- In Filename prefix, you will want to specify the name of the label template.
- In Number of instances, you will most likely want a single instance of the label template to be created. Note that based on the number of instances specified, the value specified in "Filename prefix" will be used as a template for generating multiple instances.
- In Open first instance in editor, you will most likely want to check the box so that the label template will be displayed in Oxygen (when created).
- In Default Namespace, the value should be auto-populated with the value specified in the masterSchema. The value should be fairly close to the value displayed above.
- In Prefix and Namespace, the values should be auto-populated and should be fairly close to the values displayed above.

Once you have verified the above values, select the "Options" tab (at the top of the dialog box).

- In Generate optional elements, you will most likely want to check the box so all optional elements are generated in the label template.
- In Generate optional attributes, you will most likely want to check the box so all optional attributes are generated in the label template.
- In Values of elements and attributes, you will most likely want to select "Default" as the option.
- In Preferred number of repetitions, you will most likely want to specify "1" so that a single instance of all elements and attributes will be generated in the label template.
- In Maximum recursivity level, you will most likely want to specify "1" as the maximum allowed depth in case of recusivity.
- In Choice strategy, you will most likely want to specify "Random" values over "First" value in series.

Once, you verified the above values, click on the "OK" button. This will generate the label template.


Ingest_DD_instance1.xml [D:\WINWORD\aaaSysEng\PDS4_DataModel\Data_Prep_HandBook\aaaVer_8_20111011\schemas\Schema_0700j_120403\Ingest_DD_instance1.xml] - <oX

Ingest_DD "http://pds.nasa.gov/schema/pds4/pds/
- full_name full_name0
- local_identifier local_identifier0
- comment comment0
- last_modification_date_time 0000-01-01T00:0
- pds4_merge_flag false
- DD_Attribute name0
- DD_Class name3

```
<?xml version="1.0" encoding="UTF-8"?>
<Ingest_DD xmlns="http://pds.nasa.gov/schema/pds4/pds/v07"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v07 file:/D:/WINWORD/aaaSysEng/PDS4_DataM
    <full_name>full_name0</full_name>
    <local_identifier>local_identifier0</local_identifier>
    <comment>comment0</comment>
    <last_modification_date_time>0000-01-01T00:00</last_modification_date_time>
    <pds4_merge_flag>false</pds4_merge_flag>
    <DD_Attribute>
        <name>name0</name>
        <version_id>version_id0</version_id>
        <local_identifier>local_identifier1</local_identifier>
        <steward_id>steward_id0</steward_id>
        <name_space_id>name_space_id0</name_space_id>
        <submitter_id>submitter_id0</submitter_id>
        <definition>definition0</definition>
        <comment>comment1</comment>
        <Terminological_Entry>
            <name>name1</name>
            <definition>definition1</definition>
            <language>language0</language>
            <preferred_flag>false</preferred_flag>
            <External_Reference_Extended>
                <reference_text>reference_text0</reference_text>
                <description>description0</description>
                <name>name2</name>
                <url>http://www.oxygenxml.com/</url>
            </External_Reference_Extended>
        </Terminological_Entry>
        <DD_Value_Domain>
            <enumeration_flag>false</enumeration_flag>
            <value_data_type>value_data_type0</value_data_type>
            <formation_rule>formation_rule0</formation_rule>
            <minimum_characters>minimum_characters0</minimum_characters>
            <maximum_characters>maximum_characters0</maximum_characters>
            <minimum_value>minimum_value0</minimum_value>
            <maximum_value>maximum_value0</maximum_value>
            <pattern>pattern0</pattern>
```

From the above step, you should have successfully created a label-template for the Ingest_DD product.   With the label-template visible, ensure that in the top right, you can see a little green box. If the box is red (or not green), the label-template (xml document) is not valid. As the sample label was generated by the XML editor, there shouldn't be any errors. Contact your PDS rep to resolve any discrepancies.

The label-template when populated will become the data-dictionary label.

Step #4: Examine the as-yet-unedited data dictionary label in your favorite XML-aware editor. You are now ready to begin populating the data dictionary label with the metadata associated with each attribute and class that is to be defined in the local data dictionary.

As you populate the dictionary label, ensure that the XML is fully formed (i.e., the XML editor will validate the XML and will have an indicator (which is usually a green or red box) that indicates if errors are present in the XML). Or, you can use the PDS4 Validation Tool to further validate the contents of the XML label.

Step #5: At this point, you should have a fully formed (populated) compliant XML data dictionary label. A decision point has been reached whereby the contents of the locally defined data dictionary may be "merged" with the PDS4 Data Dictionary (via ingestion into the PDS4 Information Model).

To "merge" the locally defined data dictionary with the PDS4 Data Dictionary, set the "pds4_merge_flag" to "true". If you choose not to "merge", then set the "pds4_merge_flag" to "false". Save all changes to the data dictionary label file.

Step #6: In the future, you will eventually access the online Data Dictionary Service, "point" the service to your data dictionary label, and press "go". The online Data Dictionary Service is under development.

So, the workaround is to email the data dictionary label file to the SE node for processing. The SE data engineer will process your file and create a local instance of the "Node & Mission" schemata (XSD) and a report describing any anomalies encountered while translating / merging.

If you have elected to "merge" the locally defined data dictionary with the PDS4 Data Dictionary, the "service" will additionally validate the contents to ensure a compliant "merge" of the elements and classes is possible (e.g., no duplication of element / class names within the same Registration Authority, etc.). If the SE data engineer doesn't detect any anomalies, then the data engineer will "register" the contents of your data dictionary label with the PDS4 Information Model. This will ensure that your locally defined attributes and/or classes will be contained in the next build of the PDS4 Data Dictionary.

The end result is that you will have online access to the node and mission schemas (xsd).

Step #7: Now that you have a valid local instance of the node and mission schemas (xsd), you can incorporate these dictionaries into your data product pipeline. This is done through an XML include reference in your specific product schema. So, the next step is to "link" the set of node and mission schemas (that are to be referenced) into the product schema.

There are four types of references:

- A reference to a schema that is in the same directory as the label-template
- A reference to a schema that is locally-cached / defined
- A reference to a schema that is online accessible
- A reference to a schema using an XML-catalog

In all of the examples that follow, these equivalences are made:

a.	the "pds" namespace to:	`"http://pds.nasa.gov/pds4/pds/v08"`
b.	the "phxmd" namespace to:	`"http://pds.nasa.gov/schema/pds4/phxmd/v01"`
c.	the "phxmd" schema file:	`"phxmd.xsd"`

1. Here is an example of a reference where the schema and the lable are in the same directory.

```
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/phxmd/v01 file:/phxmd.xsd">
```

2. Here is an example of a reference where the schema is locally defined.

```
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/phxmd/v01
  file:/D:/schemas/phxmd.xsd">
```

3. Here is an example of a reference where the schema is online accessible.

```
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/phxmd/v01
  http://pds.nasa.gov/schemas/phxmd.xsd">
```

4. Here is an example of a reference where the schema is referenced using an XML-catalog. For more information on using XML-catalogs, consult the appendix on XML Catalog references.

```
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/phxmd/v01 junk.xsd">
```

Step #8: Now that you have all of the pieces in place, you can incorporate all of these files into your data product pipeline that will pump out gazillions of PDS compliant labels --- don't forget to validate, and then validate again, and again, and again…

11.2 Ingest_DD – Attribute Definitions

The following is a summary of the attributes and associated definitions that are used in Ingest_DD:

- local_identifier - This is used to identify the Ingest_DD. It is not currently used but is reserved for future capabilities.
- comment - This provides a local note about the Ingest_DD. It it not ingested into the data dictionary.
- full_name - This provides the name of the individual who takes responsibility for this local dictionary.
- last_modification_date_time - This provides the date and time of the last modification to this template.
- pds4_merge_flag - This indicates whether or not the Ingest_DD is to be merged into the PDS4 Data Dictionary Data Base (DDDB)
- DD_Attribute- The attribute class is repeated for each attribute defined.
    - local_identifier - This is used to uniquely identify the attribute within this template. It is used to associate this attribute to one or more classes defined within this Ingest_DD. Local_identifier is not ingested into the PDS4 DDDB.
    - comment - This provides a local note about this attribute. It is not ingested into the PDS4 data dictionary.
    - version_id - This provides the version of the attribute.
    - name - This provides the title of the attribute and is the name provided in the preferred terminological entry.
    - steward_id - This provides the steward of the attribute.
    - name_space_id - This provides the name space identifier for the attribute and is assigned by the steward.
    - submitter_id - This provides the name of the author that submits the item to the steward.
    - definition - This provides the definition of the attribute and is the definition provided in the preferred terminological entry.
- DD_Value_Domain- The value domain exists once per per attribute definition.
    - enumeration_flag - This indicates whether the attribute has permissible values defined.
    - formation_rule - This provides a guide as to how the value of the attribute is to be formed, for example hh:mm:ss for time.

    - minimum_characters - This provides the minimum number of characters allowed for an attribute with a character data type. If no value is provided then the minimum characters of the character data type is used.
    - maximum_characters - This provides the maximum number of characters allowed for an attribute with a character data type. If no value is provided, then the maximum characters of the character data type is used.
    - minimum_value - This provides the minimum value for an attribute with a numeric data type. If no value is provided then the minimum value of the numeric data type is used.
    - maximum_value - This provides the maximum value for an attribute with a numeric data type. If no value is provided then the maximum value of the numeric data type is used.
    - pattern - This provides a rule for the validation of a permissible value.
    - unit_of_measure_type - This provides the set of units of measure allowed.
    - specified_unit_id - This provides the unit with respect to the minimum and maximum values.
    - value_data_type - This provides the data type of the attribute. If the attribute previously exists in the DDDB then the DDDB data type is used.
- DD_Permissible_Value- The permissible value class is repeated for each enumerated value.
    - value - This provides a permissible value.
    - value_meaning - This provides the meaning of the permissible value.
    - value_begin_date - This is the date from which a permissible value is valid within the PDS4 DDDB,
    - value_end_date - This is the date to which a permissible value is valid within the PDS4 DDDB.
- Terminological_EntryThis class is repeated for each unique name and definition of the attribute.
    - name - This is a title of the attribute.
    - definition - This is a definition of the attribute.
    - language - This provides the language in which the name and definition are written.
    - preferred_flag - This indicates which name and value meaning are preferred.
- Bibliographic_Reference- This class is repeated for each reference associated with a terminological entry.
    - name - This provides the title of the referenced artifact.
    - description - This provides the rationale for the bibliographic reference.
    - doi - This provides a document object identifier (doi) a document referenced by this terminological entry.
    - reference_text - This is a formal bibliographic reference for this terminological entry.
    - url - This provides a url to a website used as a reference for this terminological entry.
- DD_Class- This is used to define the PDS4 class.
    - local_identifier - This provides a local identifier for the class.
    - version_id - This provides a version identifier for the class.

  - name - This provides a title for the class.
  - steward_id - This provides a steward for the class.
  - name_space_id - This provides a name space identifier for the class. It is assigned by the steward.
  - definition - This provides a description of the class.
- DD_Association- The association class is repeated for each attribute to be associated to this class.
  - local_identifier - This identifies the attribute or class to be associated.
  - constant_value - This provides the value to be used if an attribute is static.
  - maximum_occurrences - This provides the maximum number of times the attribute is repeated.
  - minimum_occurrences - This provides the minimum number of times the attribute is repeated.
  - reference_association_type - This provides the name of the reference. The name 'has_attribute' is used for attribute associations. The name "extension_of", "restriction_of" or "subclass_of" is used for class associations.

11.3 Example Ingest_DD Product Label

For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of a Ingest_DD product label, consult the PDS4 Data Dictionary.

An unpopulated sample Ingest_DD.xml template is available here and a completed Ingest_DD for the Imaging Telemetry_Parameters class is available here.

PART V. GENERATE THE ARCHIVE

12.0 LABELS, INVENTORIES, PACKAGES

[TBD]

13.0 VALIDATION

This section describes the process of validating the object-oriented design and the inherent relationships of and between the generic schema, the specific node and mission schema, schematron asserts, and the resulting child XML document,

Figure 13-1 illustrates the process by which users ensure the resulting XML documents are compliant to the parent schemas. The validation process guarantees the object-oriented design of the parent-child relationships are preserved through out the design and implementation stages of preparing XML documents; specifically that:

- The PDS4 product labels validate/are valid against all of the following:
  a. the generic master-schema
  b. the specific node and mission schema; if provided
  c. schematron file(s); if provided.

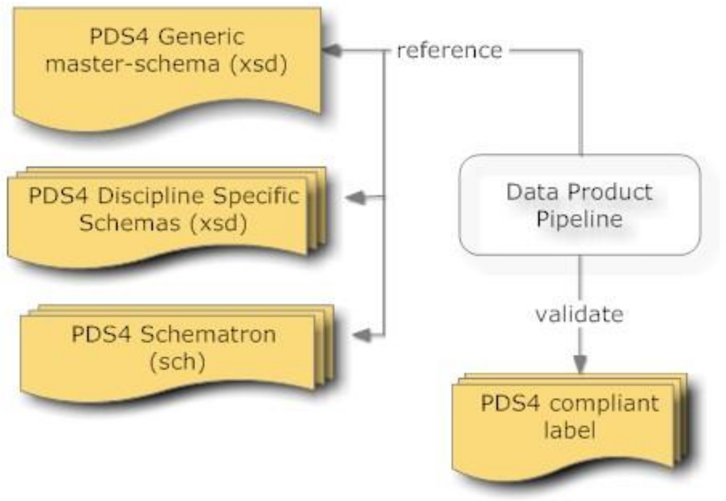


Figure 13-1. Diagram of the Validation of a Product Label

Your Data Product Pipeline will need to incorporate a mechanism by which the pipeline can validate the resulting PDS4 product labels against the noted schema and schematron files.

13.1 PDS Validation Tool

PDS provides a validate software tool which can be found here:

http://pds.nasa.gov/pds4/software/validate/

PART VI. SAMPLES, EXAMPLES, OTHER TUTORIALS

14.0 EXAMPLE PDS4 PRODUCTS

A PDS4 tutorial would not be complete without providing a set of PDS4 products that were generated from example PDS3 products.

There are two different sets of examples:

- The first is a set of example products. This includes a representative set of products that would exist within an archive (e.g., character table, binary table, document, etc.).

- The second set is a complete archive. This includes all products that would comprise a complete PDS4 archive (e.g., bundle, collections, collection inventories, readme, errata, and basic products). The archive is presented in a directory structure required of an archive.

  This material was originally archived under PDS and is representative of a PDS3 dataset migrated to become a PDS4 "example" archive. Consequently, the products within the "example" archive use "identifiers" that identify the products as "example.DPH" products (i.e., the "identifiers" are not representative of what an actual science archive would use).

  This is intentional and allows these products to be registered with the PDS registry service and searchable as "example products" and "example archive".

14.1 PDS4 Product Examples

The set of PDS4 product examples can be found at:

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_8k/

Within this set of examples, there is an example of the following product types:

1. ARRAY_2D_IMAGE extension of the PDS4 Array_Base, (i.e., Homogeneous N-dimensional array of Scalars) class where a contiguous stream of BINARY data, assembled as a two dimensional data structure, maps the "items" contained in a ARRAY_2D_IMAGE file.

2. TABLE_CHARACTER extension of the PDS4 Table_Base (i.e., Heterogeneous repeating record of Scalars) class where a contiguous stream of ASCII characters, assembled as fixed-width fields, maps the "items" contained in a TABLE_CHARACTER file.

3. TABLE_CHARACTER_GROUPED extension of the PDS4 Table_Base (i.e., Heterogeneous repeating record of Scalars) class where a contiguous stream of ASCII characters, assembled as sets of repeating fixed-width fields, maps the "items" contained in a TABLE_CHARACTER_GROUPED file.

4. TABLE_BINARY extension of the PDS4 Table_Base (i.e., Heterogeneous repeating record of Scalars) class where a contiguous stream of BINARY data, assembled as fixed-width fields, maps the "items" contained in a TABLE_BINARY file.

5. TABLE_BINARY_PACKED extension of the PDS4 Table_Base (i.e., Heterogeneous repeating record of Scalars) class where a contiguous stream of BINARY data, assembled as fixed-width fields, maps “Packed_Data_Fields” and “Field_Bit” classes contained in a TABLE_BINARY file.

6. DELIMITED_TABLE class where a contiguous stream of ASCII characters, combined with a field_delimiter and record_delimiter scheme, maps the "items" contained in a CSV “like” file.

7. DOCUMENT class where one or more instantiations of a document (e.g., ascii text, pdf, html), as identified as a set, comprise a logically complete “copy” of the referenced document product.

8. HEADER and TABLE_CHARACTER classes illustrating how combined digital objects can co-exist in a single data product file.

The files that comprise the PDS4 example products can be found at:

- PDS4 migrated data_set files (directory listing):

  http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_8k/dph_example_products/

A zip file of the PDS4 migrated data can be downloaded from:

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_8k/zip_ExampleProducts.zip

14.2 PDS4 Example Archive

This set of PDS4 examples were derived from a PDS3 PPI data_set:

```
DATA_SET_ID = "VG2-J-PLS-5-SUMM-ELE-MOM-96.0SEC-V1.0"
```

Both the original PDS3 data_set files and the equivalent PDS4 product files are presented for the purposes of illustrating (comparatively) how a PDS3 data_set can be migrated to PDS4.

14.2.1 PDS3 Data_set Files

The directory structure of the original PDS3 PPI data_set consisted of the following:

```
-  aareadme.txt
-  checksums.txt
-  errata.txt
-  \browse
-  \catalog
-  \data
-  \documents
-  \documents\mission
-  \documents\pls
-  \documents\symbols
```

The files that comprise the original PDS3 PPI data_set can be found at:

- PDS3 PPI data_set files (directory listing):

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_8k/dph_example_PDS3_VG2PLS/

A zip file of the PDS3 VG2PLS data_set can be downloaded from:

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_8k/dph_example_PDS3_VG2PLS/
zip_VG2PLS_archive.zip

14.2.2 PDS4 Product Files

The PDS4 archive structure, as migrated from the above PDS3 files, consists of the following:

```
-  Product_Bundle.xml
-  README.TXT
-  \browse
-  \context
-  \data
```

- `\document`
- `\schemas`
- `\local_dictionaries`

The following PDS3 files have been relocated to the “document” directory:

- `checksums.txt`
- `errata.txt`

The files that comprise the PDS4 migrated data_set can be found at:

- PDS4 migrated data_set files (directory listing):

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_8k/dph_example_archive_VG2PLS/

A zip file of the PDS4 migrated data can be downloaded from:

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_8k/dph_example_archive_VG2PLS/zip_VG2PLS_archive.zip

15.0 OTHER TUTORIAL MATERIAL

Additional tutorial material can be found on:

TBD

APPENDIX A ACRONYMS

The following acronyms are pertain to this document:

ADM	Architecture Development Method
API	Application Programming Interface
COTS	Commercial Off-The-Shelf
EN	Engineering Node (PDS)
ESDIS	Earth Science Data and Information System
FTP	File Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IPDA	International Planetary Data Alliance
IT	Information Technology
JPL	Jet Propulsion Laboratory
NASA	National Aeronautics and Space Administration
NSSDC	National Space Science Data Center
PDS	Planetary Data System
RM-ODP	Reference Model of Open Distributed Processing
RSS	Really Simple Syndication
SDSC	San Diego Supercomputing Center
SOA	Service-Oriented Architecture
TB	Terabyte
TOGAF	The Open Group Architecture Framework
XML	eXtensible Markup Language

APPENDIX B XML REFERENCES

[TBD]

APPENDIX C SCHEMATRON REFERENCES

In markup languages, **Schematron** is a rule-based validation language for making assertions about the presence or absence of patterns in XML trees. It is a structural schema language expressed in XML using a small number of elements and XPath.

Schematron is capable of expressing constraints in ways that other XML schema languages like XML Schema and DTD cannot. For example, it can require that the content of an element be controlled by one of its siblings. Or it can request or require that the root element, regardless of what element that is, must have specific attributes. Schematron can also specify required relationships between multiple XML files.

Constraints and content rules may be associated with "plain-English" validation error messages, allowing translation of numeric Schematron error codes into meaningful user error messages.

An example of a schematron "rule" follows:

```
    <sch:pattern>
      <sch:rule context="pds:Identification_Area">
      <!-- =================================================== -->
      <!-- Test: ensure 'information_model_version' value      -->
      <!--       matches expected-value                        -->
      <!-- =================================================== -->
      <sch:assert test="pds:information_model_version='0.8.0.0.k'">
        Identification_Area.information_model_version: does NOT specify
        the current version.
      </sch:assert>
   </sch:rule>
</sch:pattern>
```

- A set of example schematron assert statements with the files that can be used to test the validation results can be found here:

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_7j/dph_example_schematron/

- A primer to schematron is available on the PDS Wiki:

https://oodt.jpl.nasa.gov/wiki/display/pdscollaboration/Validation+of+PDS4+XML+Labels

- A good set of introductory tutorials are available here. There is no need to run through all of the tutorials. But Two Types of XML Validation, Usage and Features, and Overview will set the stage of what Schematron is :
    - http://www.xfront.com/schematron/index.html

- Great set of starter examples and how to:

    - http://zvon.org/xxl/SchematronTutorial/General/toc.html

- A walk through of schematron:
    - http://www.ibm.com/developerworks/xml/tutorials/x-schematron/

APPENDIX D XML CATALOG REFERENCES

XML documents typically refer to external entities, for example schemas for the Document Type Definition. These external relationships are expressed using URIs, typically as URLs.

However, if they are absolute URLs, they only work when your network can reach them. Relying on remote resources makes XML processing susceptible to both planned and unplanned network downtime.

Conversely, if they are relative URLs, they're only useful in the context where they were initially created. For example, the URL "../../xml/dtd/docbookx.xml" will usually only be useful in very limited circumstances.

One way to avoid these problems is to use an entity resolver or a URI Resolver. A resolver can examine the URIs of the resources being requested and determine how best to satisfy those requests. The XML catalog is a document describing a mapping between external entity references and locally-cached equivalents.

The following is an example of a XML catalog that resolves the locations of schemas as the schemas follow a somewhat natural development cycle of: DEV, ARCHIVE, and ONLINE.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <!-- ======================================================= -->
  <!-- 1st reference is to DEV instance                        -->
  <!-- ======================================================= -->
    <group xml:base="file:/D:/DEV/schemas/">
    <uri name="http://pds.nasa.gov/pds4/pds/v08" uri="PDS4_PDS_0800k.xsd"/>
    <uri name="http://pds.nasa.gov/schema/pds4/phxmd/v01" uri="PHXMD_0700j.xsd"/>
    <system systemId="PDS4_PDS_0800k.xsd" uri="PDS4_PDS_0800k.xsd"/>
  </group>

  <!-- ======================================================= -->
  <!-- 2nd reference is to ARCHIVE instance                    -->
  <!-- ======================================================= -->
  <group xml:base="file:/D:/ARCHIVE/schemas/">
    <uri name="http://pds.nasa.gov/pds4/pds/v08" uri="PDS4_PDS_0800k.xsd"/>
    <uri name="http://pds.nasa.gov/schema/pds4/phxmd/v01" uri="PHXMD_0700j.xsd"/>
    <system systemId="PDS4_PDS_0800k.xsd" uri="PDS4_PDS_0800k.xsd"/>
  </group>

  <!-- ======================================================= -->
  <!-- 3rd reference is to ONLINE instance                     -->
  <!-- ======================================================= -->
  <system systemId="http://pds.nasa.gov/schema/pds4/phxmd/v01"
uri="http://pds.jpl.nasa.gov/schemas/PHXMD_0700j.xsd"/>
  <uri name="http://pds.nasa.gov/schema/pds4/phxmd/v01"
uri="http://pds.jpl.nasa.gov/schemas/PHXMD_0700j.xsd"/>

</catalog>
```

Once the XML catalog has been created, the catalog file will need to be "registered" with the XML-aware editor that you are using. Once registered, you will also need to validate that the XML-catalog is being refereneced by your XML product labels.

- A primer to XML Catalogs is available on the PDS Wiki:

  https://oodt.jpl.nasa.gov/wiki/display/pdscollaboration/XML+Catalogs

The primer will take you through the following steps:

1. *Example Schema / XML Labels)*
2. *XML Catalog Solution, Part 1: schemaLocation Changes*
3. *XML Catalog Solution, Part 2: Creating an XML Catalog File*
4. *XML Catalog Solution, Part 3: Configuring the XML Catalog*
5. *XML Catalog Solution, Part 4: Validating the XML document using the XML Catalog*
6. *XML Catalog Solution, Part 5: Using XML Catalog to make schemas Portable*

APPENDIX E STEPS TO CREATE A LABEL TEMPLATE

This appendix illustrates the steps that can be used to create a label-template from the master-schema (xsd).

Steps:

1. Download a copy of the current masterSchema (xsd)
2. Open the masterSchema in xml-aware editor (XAE)
3. Customize the settings
4. Create label template (xml)
5. Ensure label template is valid

Step 1: Download a copy of the current masterSchema (xsd)

The current version of the PDS4 masterSchema (xsd) can always be found here:

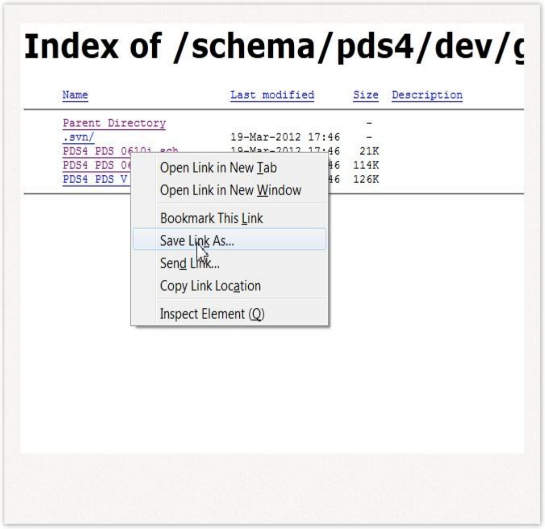
http://pds.nasa.gov/schema/pds4/current/generic/common/

The two files of interest are:

- The current schematron file: PDS4_PDS_xxx.sch
- The current masterSchema file: PDS4_PDS_xxx.sch

You will want to copy both of the above files to a local directory / folder.

- Open your favorite browser
- Navigate to the above url
- Select the file of choice to download
- Put your mouse-cursor over the file to be copied
    - right-click and select "Save Link-As" – this will open a dialog box to specify a directory / folder where you want to copy the file
    - repeat for the 2nd file



Index of /schema/pds4/dev/g

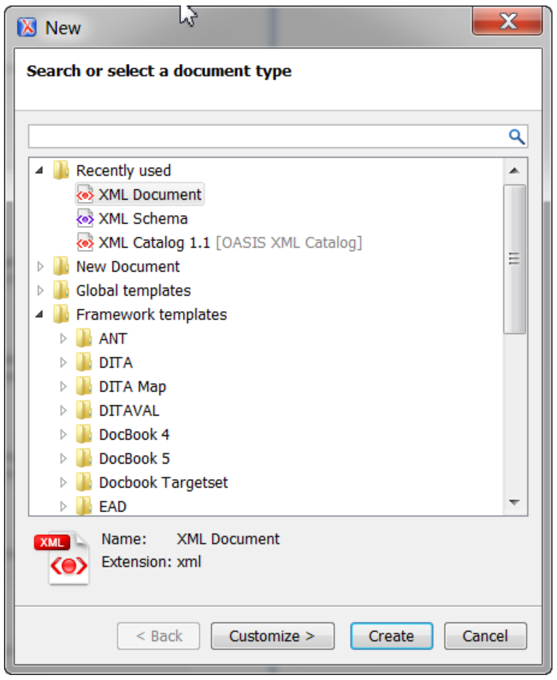
```
Name                    Last modified      Size  Description
Parent Directory                             -
.svn/                   19-Mar-2012 17:46    -
PDS4 PDS 0610i.sch      19-Mar-2012 17:46   21K
PDS4 PDS 06                             46  114K
PDS4 PDS V                              46  126K
```

Step 2: Open the masterSchema in XML-aware Editor

From Step 1, you should have a local copy of the masterSchema (xsd) and the schematron (sch) files.

Using your favorite xml-aware editor (XAE), open the masterSchema (xsd) file.

The next step is to identify the product for which you want to create a label template. The XAE The next step is to the XAE to create a new document. You will want to create a new “XML Document”.

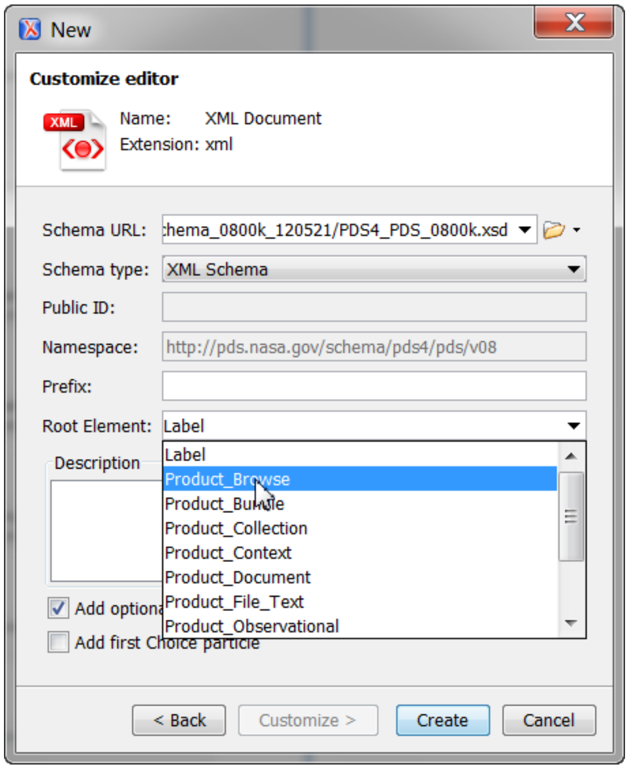


Name: XML Document
Extension: xml

Step 3: Customize the Settings

Before you create the document you we will need to “customize” some of the information that will be needed to create the label-template (xml) document that will suit your purpose.

The masterSchema includes references to a number of "products"each of which represents a distinct set of product classes. Product classes are, by definition, the highest level classification of the product into a category like "observational product" or "document":



Name: XML Document
Extension: xml

Schema URL: :hema_0800k_120521/PDS4_PDS_0800k.xsd
Schema type: XML Schema
Public ID:
Namespace: http://pds.nasa.gov/schema/pds4/pds/v08
Prefix:
Root Element: Label

Label
Product_Browse
Product_Bundle
Product_Collection
Product_Context
Product_Document
Product_File_Text
Product_Observational

Description

☑ Add optional content
☐ Add first Choice particle

1. We need to populate the 'Schema URL" with the full path to the masterSchema. Either drag-and-drop the masterSchema (xsd) or type the path.
2. For 'Schema type', select 'XML Schema' as the type of schema being referenced.
3. The 'Namespace' value should be pre-populated with the namespace identified in the masterSchema.
4. For 'Root Element' select the type of product that you want to create (e.g., Product_Browse, Product_Observational, etc).
5. You will most likely want optional content to be added when the label-template is created, so select 'Add optional content'.

Step 4: Create the label-template (xml) document

You are now ready to have the XAE create the label-template:

1. Click on 'Create'.
2. The label-template will auto-display in the XAE.
3. You will probably want to 'Save' the file locally.

Step 5: Ensure the label-template is valid (or not)

From Step 4, you should have a local copy of the label-template (xml) document. The XAE provides a simple visual indication of whether or not the XML file is valid.

The label-template that we generated from Step 4 will most likely not be valid as there are no values specified within the xml tags. Once a full set of values has been specified, the label-template should become valid:

1. With the label-template visible in the XAE, ensure that in the top right, you can see a little green box. If the box is red (or not green), then label-template (xml document) is not valid.